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.
- 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. 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.
- 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

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)

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$1_B$</td>
<td>$10_B$</td>
<td>$11_B$</td>
<td>$100_B$</td>
<td>$110_B$</td>
<td>$2_B$</td>
<td>$20_B$</td>
<td>$21_B$</td>
<td>$22_B$</td>
<td>$12_B$</td>
<td>$B_B$</td>
<td>$B1_B$</td>
</tr>
</tbody>
</table>

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: __________

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

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”.

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”.  

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 0 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
```
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...

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

In “text your friend” format, this would be:

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

In “text your friend” format, this would be:

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

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

```
box 256 1
box 256 2
box 256 3
box 256 4
```
Question 9: I should have made a left turn at Albequerque... (8 pts) 

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.

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

Race condition
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...

<table>
<thead>
<tr>
<th>Cards in hand (top-to-bottom)</th>
<th>Table</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td></td>
<td>Starting setup</td>
</tr>
<tr>
<td>B C D</td>
<td>A</td>
<td>Put the top card on the table</td>
</tr>
<tr>
<td>C D B</td>
<td>A</td>
<td>Put the top card on the bottom</td>
</tr>
<tr>
<td>D B</td>
<td>A C</td>
<td>Put the top card on the table</td>
</tr>
<tr>
<td>B D</td>
<td>A C</td>
<td>Put the top card on the bottom</td>
</tr>
<tr>
<td>D</td>
<td>A C B</td>
<td>Put the top card on the table</td>
</tr>
<tr>
<td>D</td>
<td>A C B</td>
<td>No change, only one card</td>
</tr>
<tr>
<td></td>
<td>A C B D</td>
<td>Put the top card on the table</td>
</tr>
</tbody>
</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**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>2</td>
<td>10</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>9</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Cards in hand (top-to-bottom)</th>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCDEFGHIJ</td>
<td></td>
</tr>
<tr>
<td>BCDEFGHIJ</td>
<td>A</td>
</tr>
<tr>
<td>CDEFGHIJB</td>
<td>A</td>
</tr>
<tr>
<td>DEFHIJB</td>
<td>AC</td>
</tr>
<tr>
<td>EFHJBD</td>
<td>AC</td>
</tr>
<tr>
<td>FGHIJBD</td>
<td>ACE</td>
</tr>
<tr>
<td>GHIJBD</td>
<td>ACE</td>
</tr>
<tr>
<td>HIJBDF</td>
<td>ACEG</td>
</tr>
<tr>
<td>IJBDHF</td>
<td>ACEG</td>
</tr>
<tr>
<td>JBDHF</td>
<td>ACEGI</td>
</tr>
<tr>
<td>BDFHJ</td>
<td>ACEGI</td>
</tr>
<tr>
<td>DFHJ</td>
<td>ACEGIB</td>
</tr>
<tr>
<td>FHDJ</td>
<td>ACEGIB</td>
</tr>
<tr>
<td>HJD</td>
<td>ACEGIBF</td>
</tr>
<tr>
<td>JDH</td>
<td>ACEGIBF</td>
</tr>
<tr>
<td>DH</td>
<td>ACEGIBFJ</td>
</tr>
<tr>
<td>HD</td>
<td>ACEGIBFJ</td>
</tr>
<tr>
<td>D</td>
<td>ACEGIBFJH</td>
</tr>
<tr>
<td>D</td>
<td>ACEGIBFJH</td>
</tr>
</tbody>
</table>

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)

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)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musical Chairs</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Two Circles</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>One at a Time</td>
<td>●</td>
<td>○</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musical Chairs</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Two Circles</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>One at a Time</td>
<td>●</td>
<td>○</td>
</tr>
</tbody>
</table>

If N=6 as in diagram, A will only see 1,3,5.

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., $N^6$ is bigger than cubic, so pick exponential. (select ONE per row)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Constant</th>
<th>Logarithmic</th>
<th>Linear</th>
<th>Quadratic</th>
<th>Cubic</th>
<th>Exponential</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musical Chairs</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>$N^2$ iterations, $N$ comparisons each</td>
</tr>
<tr>
<td>Two Circles</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>$N$ iterations, $N$ comparisons each</td>
</tr>
<tr>
<td>One at a Time</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>$N$ Cal students, $N$ comparisons each</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Constant</th>
<th>Logarithmic</th>
<th>Linear</th>
<th>Quadratic</th>
<th>Cubic</th>
<th>Exponential</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Musical Chairs</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>$N^2$ iterations</td>
</tr>
<tr>
<td>Two Circles</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>$N$ iterations</td>
</tr>
<tr>
<td>One at a Time</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>$N^2$ comparisons, each one at a time</td>
</tr>
</tbody>
</table>

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).
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you apply not to each of (T F T), you get (F T F)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>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).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>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 output.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>If you keep only the elements that are not not true, that means they are true, so you keep only the trues.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Two cascaded nots means the same thing as flipping the light switch twice, so the answer is the same as two above.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keep not keeps only the false elements (F), and map of not flips their state so you get (T)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>If you flip all the switches they become (F T F), and if you keep only the false ones you get (F F)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If you apply not to each of (T F T), you get (F T F)
Question 14: Berkeley Python Flying Circus... (10 pts = 5 * 2 pts)

>>> R = range(10,20)
>>> list(R[2:4])
[2, 3]  ○ [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]  ○ [4]  ○ ["51"]  ○ ["51"]  ○ 4  ○ 51  ○ "4"  ○ "51"  ○ "31"  ● "501"  ○ Error
Lists are 0-indexed 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:

<table>
<thead>
<tr>
<th>-460º</th>
<th>40º</th>
<th>60º</th>
<th>80º</th>
<th>212º</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cold</strong></td>
<td><strong>Cool</strong></td>
<td><strong>Nice</strong></td>
<td><strong>Hot</strong></td>
<td></td>
</tr>
</tbody>
</table>

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!

```python
>>> 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).

```python
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 key_low <= key < key_high

            return SD[range]

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

>>> SD_get(sd, 1000)
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.

```python
>>> SD_get(sd, 1000)
ERROR: no range contains key!
```
c) Now let’s fix the last assumption. Fill in the blank of a new \texttt{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.

```python
>>> 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