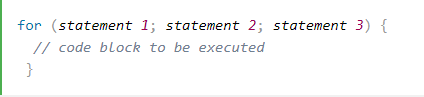
**1.** How many times is line (5) executed in the following pseudocode?

1. n = 10
2. m = 20
3. for *i* = 1 to *n* + 4
4. for *j* = 1 to *m*
5. print (*i*, *j*)

Explanation in terms of coding

This is a for loop. A for loop’s purpose is to execute the code block (the action scripted to be taken) using three statements in Java with this format:



Statement 1 is executed *one time* before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed *every time* after the code block has been executed.

In the problem, the for loop starts in line 3. The previous lines 1 and 2 define the variables, *m* and *n*. In the for loop, statement 1 defines what the counter (index in this case) is, which is *i*. It is executed once (and therefore is the initial value).

Statement 2, in Java, would state something like *i<=n+4* because the value you want must be no more than *n+4*. This part of the code is basically a range of what values *i* can be, so from *i* to *n+4*, including the values *i* (because statement 1 is executed once) and *n+4* (because the code says it must be up to that value).

* It is important to understand that the value of this range is how many times *i* can occur.
* It is also essential to note that Statement 2 defines a condition for the loop to run. If this condition is true, the loop will start over again. If it is false, the loop will end.
* In the problem, if *i* is looped enough times, then there eventually will be a time where the condition is not met (where the *i* becomes larger than *n+4* and therefore outside of the range). The loop will stop.

Statement 3, in Java, would say *i++* to demonstrate that *i* increases a value (in this case, by 1) each time the code block has been executed. This part is not displayed in the pseudocode, but it is assumed to increase by 1 each time.

* If *i* were to be increased by more than 1, statement 3 would look like this: *i=i+2*.

In this example, there are two for loops. The code block calls for a pair of *i* and *j* to be made, and when the program is run, the output should come out with both of them, kind of like coordinates.

* Each output that’s printed out should have both *i* and *j.* So, this means that both loops must be completed in order for the code block to executed (both conditions in their respective statement 2’s must be met).
* If *i*’s condition is not met, then the code block will not be executed. This goes for *j* too.

So, you can run this in a program and count how many pairs you get, which will tell you how many loops were made.

Explanation in DS

Just as Example 2.1 in Lecture 1, you can construct a 2D grid with the values (*i*, *j*). This grid can prove the Product Principle, which states that in subsequent tasks (one happens after the other), if there are *m* ways to do one task and *n* ways to do another, there are *mn* ways to do both tasks.

| **(*i*, *j*)** | ***j* = 1** | ***j* = 2** | ***...*** | ***j* = m** |
| --- | --- | --- | --- | --- |
| ***i* = 1** | (1, 1) | (1, 2) | ... | (1, 20) |
| ***i* = 2** | (2, 1) | (2, 2) | ... | (2, 20) |
| **...** | ... | ... | ... | ... |
| ***i* = *n + 4*** | (14, 1) | (14, 2) | ... | (14, 20) |

*The ellipses represent all the values in between. The rightmost bottom box is the coordinate of the maximum values, which I’ll explain (n+4, m).*

You could figure out the maximum number in each of the ranges (conditions) described in the for loops. Unlike the example in the lecture, *m* and *n* have values, and these are shown in the grid. These are used to find the last row and columns of the grid.

* *i*’s maximum value is 14 (*10 + 4 = 14*)
* *j*’s maximum value is 20 (*20 = 20*)

You can take this grid as a 2D grid, where there are *m* columns and there are *n + 4* rows. However, remember in future problems, if any of the *i* or *j* counters don’t start from 1, this may be different.

* Generally speaking, the number of rows and columns varies depending on the range.

Anyway, you can find the area of this grid. Why do you find the area? Because of the Product Principle. You can find the amount of “ways” (or loops, in this case) by multiplying the number of “options” you have (which is demonstrated by the range). For each loop that occurs, one pair is created. Think of it like this: the units of this area are loops, not measuring units.



And it’s okay to do this, as all the sets are the same size.

What I mean by “size” (or as the professor calls it, cardinality) is that each row in the grid is a set. Each row has the same number of values in it. This problem has 14 rows. This is necessary because if there is a missing value in the table or one pair is excluded, then the “rectangle” of the grid is not whole, and therefore you cannot find the “area” using this way.

When solving other problems like this one with the Product Principle, make sure that the size is the same. If it isn’t, you would have to apply the Sum Principle.

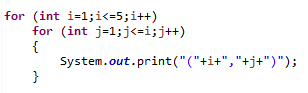
To find how many loops there are, you would multiply the two maximums (20 and 14). The amount of loops you would get is **280 total loops** (the total “S”ize of all the sets).

**2.** How many times is line (3) executed in the following pseudocode? Express your answer as an efficient formula in terms of *n*.

1. for *i* = 1 to 5*n*
2. for *j* = 1 to *i*
3. print (*i*, *j*)

This problem asks for the answer in terms of a formula of how many times the codeblock is executed. This is because *n* is not defined. Later, when you come up with a formula, you can test it out by giving a value for *n* and seeing if it makes sense.

Right now, here is what the code looks like in Eclipse *without* the variable *n* (or in other words, if *n* was equal to 1 and thus is not included in statement 2):



When this code is executed, you get this output:



Notice the *j* values go up to match the *i* value in each pair before they move onto the next pair. Think about it: if the value of *n* were to equal 2, in this scenario, the maximum of the range would go up to 10, and the numbers would be completely different. You will use this later to prove your point.

Let’s sort these pairs into a grid concerning 5*n* instead of just 5.

| **(*i*, *j*)** | ***j =* 1** | ***j* = 2** | ***...*** | ***j =* *i* = 5*n*** |
| --- | --- | --- | --- | --- |
| ***i =* 1** | (1,1) |  | ... |  |
| ***i =* 2** | (2,1) | (2,2) | ... |  |
| **...** | ... | ... | ... | ... |
| ***i =* 5*n*** | (5*n*,1) | (5*n*,2) | ... | (5*n*, 5*n*) |

There are a couple of things to notice now that there is a visual representation of the coordinates:

* There are empty spaces in the grid. This means that the Product Principle cannot be applied. If each row is a set, the size of each set is not the same. So, this opens an opportunity for the Sum Principle to be used to create a formula.
* The sets are all mutually disjoint. The sets don’t overlap with each other. That means they don’t have any pairs in common.

The Sum Principle states that if there are *m* ways to do one task and *n* ways to do another, then both tasks can be done in *m* + *n* ways. It is also helpful to note that in Lecture 1’s notes, the Sum Principle displays adding the size of each set is equal to the total size (or in this case, number of loops performed).



So, here is where you apply the rule “divide and conquer”, as the professor calls it, to find the formula. You can start from the top in this instance.

* How many pairs are in the first row? 1. How about the row after that? 2. The number of pairs increases by 1 as it goes on, continuing on for however many are needed.
* How many pairs are in the bottom row? 5*n*. This is because the highest value *j* can be is *i* according to line 2 (*j* = *i*), and the highest value *i* can be is 5*n*. The term 5*n* is the range.

For the row after the bottom, there would be pairs. Because you noticed the pattern that the number of pairs increased by 1 as you moved down the row, the pattern would apply vice versa; the number of pairs would decrease by 1. This trend would continue upwards.

You can express these patterns in the following formula using the Sum Principle:

HOWEVER, this is not the final equation. The problem is that there are ellipses in between to represent possible terms depending on what *n* would be.

Let’s say that there *n* = 1. The whole equation would be:

The quickest way to add up the numbers would be in pairs. So, you would pair up the first and the last number, then continuing on. Notice that each sum is equal to 6.

You would get , which would be equal to 30. Since each sum is equal to 6 and there are 5 pairs you can make, you can do to also get 30.

However, also remember that these are pairs. To get pairs, you must divide the number by 2 to get the total number of times the command was executed, which is 15. Look back at the number of pairs you got as the output for Eclipse. There are 15 pairs there, too.

Using this same reasoning, let’s take a look at the formula using the Sum Principle (with the ellipses). You can add the first and last term to get . See, each pair you make would also equal this expression.

So, how do you know how many pairs are in this? You can do this by thinking about what the range of the numbers is. Looking on the grid, it is 5*n*. You can also figure this out by looking at the pseudocode again. What value does *j* go to? 5*n*, because *j* = *i*, and *i* = 5*n*.

However, remember this may not always be the case; in this instance, *j* starts from 1. There may not be 5*n* terms next time. If *j* started from 2, the number of pairs would be .

The last thing you need to do is divide the whole thing by 2. Remember, you want pairs. So, the **final formula** would be:

To check, you can test this out again by plugging in 1 to *n*. You would still get 15, which is the number of times the command was executed if *n* were to equal 1.

**3.** Determine the value of the output in the following pseudocode.

1. for *i* = 1 to 80
3. print *sum*

First, before solving this, you should find out what this pseudocode means. Although it looks confusing, everything is okay.

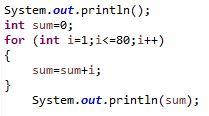
Let’s call this black box checking.

| ***sum*** | ***i*** | ***sum + i = sum*** |
| --- | --- | --- |
| 0 | 1 | 0 + 1 = 1 |

In the first row of values, the “*sum* =” is 1. The way this pseudocode works is that the “*sum* =” is the new sum for the next loop. So, in this case, 1 is the next sum. In the row after that, the new sum to be used in the “equation” part of the code is 3.

| ***sum*** | ***i*** | ***sum + i = sum*** |
| --- | --- | --- |
| 0 | 1 | 0 + 1 = 1 |
| 1 | 2 | 1 + 2 = 3 |

This continues all the way until *i* is equal to 80. Some things to note about this particular for loop pseudocode:



This is how the code looks like in Eclipse. Notice how the equation, *sum = sum + i* is inside the code block, and the command (print *sum*) is outside the brackets. This is shown through the pseudocode with the absence of indentation.

Now, for how this works. It’s all the same as previously described in problem 1, but

1. defining the value *sum* = 0 only is executed once
2. the codeblock being executed is *sum = sum + i*

For a, this is true of any integer equals above the for loop. It only happens once, so the second “*sum*” in the codeblock is NOT 0, but instead, the answer of whatever 0 + 1 is, in this case. The code will keep looping until the condition is no longer met, which is when *i* eventually equals 81.

Given that the “print *sum*” command is outside the for loop, this means that when *i* is equal to 80, whatever that final sum is, it’ll be the value that is printed (the output).

So, how would you figure out the output? An idea: using sigma notation. Why?

Given the way sigma notation works, the value of initial value *m* increases by 1 every time. This can be the *i* value as seen applied in the Java version. The *i* value must be defined. In this case, *i* starts at 1 and increases all the way to 80. In the sigma, 80 will be the final value, therefore taking that top bar.

In previous sigma notation examples, there usually is a formula in front of it that generates the value of each term in terms of *i*. Well, you have to come up with a formula for this part.

First, establish that the *i* part in the third column is the *m* + 1, *m* + 2 part in sigma notation.

Second, how do you get the formula of a sigma? You add the first and last terms together and multiply it by the number of pairs, then divide all of that by 2 because you paired it.

So, let’s make a sample sigma like this, with *n* eventually going to represent 80 in the future. Doing this works because you can set the starting value, *i*, as 1, just as it is in the pseudocode.



The sequence for this would look like this:

Add the first and last terms together, so you’ll get *n* + 1. That’s one part done. Now, for the next part, you should find how many pairs there are in this. Of course, that would be *n*, and it makes sense because the sequence’s first value is 1. Or, you can prove this using the pair expression found in problem 7 () where *m* is the initial value. is equal to *n*. Then, divide by 2, and you’ll get the following expression:



If you replace the *n* with 80, you can find the output to the pseudocode, which is **3240**. (This is also the answer in the code.)

**4.** Calculate each of the following. Your answer must be a number. No arithmetic operations are allowed in your answer. Please give 7 places after your decimal point if you use scientific notation.

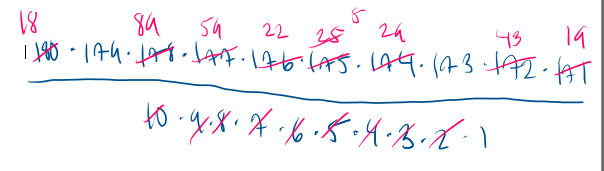
1. 
2. 
3. 

For part A, to simplify this, you need to do the same as part B. You’re going to use ellipses this time (because writing them out will take too long), so you’ll get . Cancel the factorial 690! You can write all the values out and cancel them out, but that’s really a long process and you have to check yourself multiple times.

You could have also used the nCr option in the TI-84 calculator and did 740 choose 50 since they’re both equal to each other.

The answer is **1.749801543E78**.

For part B, you can simply the factorials to . You can cancel out the 170! factorials in the denominator and numerator. From then on, you can cancel:



As you can see, many terms were cancelled out. You can multiply from this point, and the answer you’d get is **7.628275985E15**.

For part C, you can rewrite it as . Here, you’re basically algebraically distributing the denominator to the two terms on the top. You can solve each of them individually and then subtract. For the first expression, you can simplify it to and you will get when you cancel out the 667! factorial. You can simplify the second expression the same way to and cancel out the 665! factorial to get . You can subtract what you get from these two values to get the answer **446892**. The difference caused by the 2nd factorial is negligible.

**5.** Simplify the expression:



How to simplify factorials involving variables according to Chillmath:

1. Compare the factorials in the numerator and denominator
2. Expand the larger factorial such that it includes the smaller in the sequence
3. Cancel out the common factors between the numerator and denominator
4. Simplify further by multiplying or dividing the leftover expression

In this problem, the larger factorial would be the numerator. So, you would expand the numerator just as follows:

After that, you can cancel the denominator and the corresponding expression. Now, all you have left is . So, you must multiply them together to make a simplified answer: **16*n*4 - 16*n*3 - 4*n*2 + 4*n****.*

**6.** Calculate the following sum expressed in sigma notation:



Explanation w/ the TI-89 Calculator

1. Hit F3 to select the Calc menu
2. Hit the number 4 to pick the “sum” option
3. First, enter the expression in front of it (in this case, 5*k* + 8)
4. Then, comma variable *k*, comma the value *k* equals to (in this case, 1)
5. After that, comma the value at the top of the sigma (in this case, 14)
6. Close the parentheses (optional) and hit enter
7. Make sure to check the displayed sum is correct

Explanation w/ doing the work

To solve a sum, it is important to understand how sigma notation is written:



According to this, you must “divide and conquer” (like the Sum Principle). You would plug 1 into the expression for starters, find the value of that, then continue until you get to 14. Of course, this is a really long process so it’s more preferred to do using a calculator.

The answer (at least, according to the calculator) is **637**.

**7.** Find an efficient formula in terms of *m* and *n* for the following sum expressed in sigma notation. HINT: Use the handshake formula.



Let’s test out a theory and find the formula of the following sigma notation, where *i* replaces *k* as the variable:



You can figure out the expressions you need to form a formula: the number of pairs. Plug in values for *m* and *n* to test this out (5 and -1 respectively). Remember, the number of pairs is NOT *n* because *m* might be starting from an integer other than 1, so it’s important to test it out first by plugging in sample values.

You can tell by this sequence that there are 7 pairs that can be made from this sequence. You can try to make sense of this and come up with an expression for the number of pairs for this particular instance. Since the number of values depend on both *n* and *m*, they are necessary in this expression.

However, notice that since *m* is -1 in this scenario, when counting pairs, you need to consider that to get the number of pairs, you don’t want to subtract away from the *n* value. This is because *n* is 5 and needs to be increased. If *m* is -1, then in the pair expression, *m* needs to be negative. Now, you have which is equal to 6, which is not the number of pairs in this instance. You need to add 1 to get the correct number 7.

Officially, you’ll get the expression for the number of pairs. This is the most basic form of the pair expression. It can be modified depending on what *m* and *n* are. You can test it for other problems, and it’ll come out right.

So, let’s apply this to the problem. The sequence would look like this:

Now, knowing that there is an *n* in the original pair expression, you can change the pair expression to be . Why? Because you can just add the 2 to the pair expression since the final term is *n +* 2 instead of just *n*. This is one part of the formula done.

Since you have the sequence (and also because it’s in the sigma), you have the first and last term, being *n* + 2 and *m*. You can add them together to get the second part of the formula ().

Put both expressions on the top, then divide by 2 (like in the handshake formula), and the **final formula**will be:

**8.** Calculate the following product expressed in pi notation:



Explanation w/ the TI-89 Calculator

1. Hit F3 to select the Calc menu
2. Hit the number 5 to pick the “product” option
3. First, enter the expression in front of it (in this case, 3*k* + 3)
4. Then, comma variable *k*, comma the value *k* equals to (in this case, 1)
5. After that, comma the value at the top of the sigma (in this case, 9)
6. Close the parentheses (optional) and hit enter
7. Make sure to check the displayed product is correct

Explanation w/ doing the work

To solve the product, it’s important to understand how pi notation is written:



According to this, you would just plug in the numbers and evaluate. It’s a long process, plugging in 1 for starters and then going all the way up to 9. Notice the top number is usually smaller as multiplication can bring in huge numbers.

The answer (according to the calculator) is **71425670400** (or in scientific notation, **7.14257E10**).

**9.** How many three-letter “words” can be made from 5 letters (FGHIJ) if repetition of letters

1. is allowed?
2. is not allowed?

For more detailed explanations, refer to problem 14. For part A, there are three spaces for three letter words ( \_ \_ \_ ). Since there are 5 letters, there are 5 possibilities for each spot. Multiply them together () to get **125**.

For part B, your number of options available for each space decreases by 1 each time a number is selected. So, you would have to multiply this expression () to get **60**.

**10.** A president, a treasurer, and a secretary are to be chosen from a committee with 40 members. In how many ways could the three officers be chosen?

There are 40 members who are able to become president, in which 1 is chosen. Once a president is chosen, there will be 39 members who can take the role of treasurer, so now 2 are chosen from the total number of 40 (the president and the treasurer). After that, there will be 38 members who are eligible to become secretary.

Assuming that officers cannot take more than one position, the math that needs to be done is to get the total amount of ways that they can be chosen, which is **59280**.

**11.** A bit is a 0 or a 1. A bit string of length 9 is a sequence of 9 digits, all of which are either 0 or 1.

1. How many bit strings of length 9 are there?
2. How many bit strings of length 9 or less are there? (Count the empty string of length 0 also.)

For part A, there are 9 spaces. Each has 2 possibilities, which are 0 or 1. So, for this part, multiply two 9 times (29), and the answer would be **512**.

For part B, you can plug in different exponents to 2, just as long as it’s under 9. Then, you add them all up (including 512). You can create a sum (sigma notation) equation of this, like this:



In this case, the sum would be **1023**.

**12.** Find how many positive integers with exactly four decimal digits, that is, positive integers between 1000 and 9999 inclusive, have the following properties:

1. are even
2. are divisible by 5 or by 7 (inclusive or)
3. are divisible by 7
4. are divisible by 5

Before beginning any of the problems, it is important to notice that there are 9000 numbers within the range. Also, the “exactly four decimal digits”, this may be pertaining to how many places there are but this is not confirmed.

Now, for part A, for a number to be even, it must be divisible by 2. The Quotient Principle is that the task has *n* ways to do something. But, for a specific way, there has to be *d* ways for it to happen, thus getting the formula . In this case, there are 9000 integers (*n* ways). So, you divide 9000 (the size of the range) by 2 to get **4500** numbers.

As for part B, since the condition uses the inclusive “or”, you can use the subtraction rule, where if an event can occur either in *m* ways OR in *n* ways (overlapping), the number of ways the event can occur is then minus the number of ways the event can occur commonly to the two different ways. The number of *m* and *n* ways are found in parts C and D. However, you must find the minus portion.

Since you want to find the number of ways the event can occur commonly, you must take it as the “number of integers divisible by 5 AND 7”. To do this, you need to perform the Quotient Principle, and the denominator needs to be 35. Why? Because 5 times 7 ways is the number of times it can occur commonly (product). For the answer to this portion of the problem, divide 9000 by 35 to get 257.

Now all there’s left is to plug in all the values: **2829**.

In part C, for numbers that are divisible by 7, make sure to divide the 9000 by 7 to get the answer for the same reasoning as in part A, which is about **1286**. Be sure to round down for this one.

In part D, for numbers that are divisible by 5, make sure to divide the 9000 by 5 to get the answer for the same reasoning as in part A, which is **1800**.

**13.** 4 letter words are formed using the letters A, B, C, D, E, F, and G. How many such words are possible for each of the following conditions?

1. No condition is imposed
2. No letter can be repeated in a word
3. Each word must begin with the letter A
4. The letter C must be at the end
5. The second letter must be a vowel

For part A, there are no conditions imposed. Refer to the next problem’s (14) part A for a more thorough explanation. Going off of that, there are four spaces as well ( \_ \_ \_ \_ ). There are 7 possibilities (letters) for each space. So, 74 = **2401**.

In part B, no letter can be repeated. Refer to problem 10 for more elaborate reasoning on the permutation. Since no letter can be repeated, you must subtract 1 every time you use a possibility, so in this case, there are 7 for the first space, 6 for the second, and so on. Multiply them together () to get the answer **840**.

For part C and part D, there are four spaces ( \_ \_ \_ \_ ). The first or last space is taken by the value “1” as there is only 1 possibility for that space (A or C). This is different from problem 14 part C where designated spaces must be the same letters, not a specified letter like in this problem. Then, the rest of the spaces are free to be whatever letter, so therefore the three spaces have 7 possibilities each. Multiply these together (or ) and you get **343** for both parts.

In part E, the second letter must be a vowel. So, just like in the previous parts, all the other spaces are free spaces with 7 possibilities each. However, the second space requires only 2 possibilities, because the only letters that can fill it are the 2 vowels in the set. So, the equation would be and the answer would be **686**.

**14.** A fair 6 sided die is rolled 4 times and the resulting sequence of 4 numbers is recorded.

1. How many different sequences are possible?
2. How many different sequences consist entirely of even numbers?
3. How many different sequences are possible if the first, third, and fourth numbers must be the same?

Like in Mr. Cohen’s statistics class at GCIT, you have 4 empty spaces (the 4 times that were rolled), represented by \_ \_ \_ \_. To answer part A of the question, you should understand that there are 6 possibilities for each space: 1, 2, 3, 4, 5, and 6. Since there are no conditions imposed by the first part, you can multiply 6 four times (or 64 to get **1296**.

For part B, you don’t have 6 possibilities for each space because there are some odd numbers mixed in. The condition requires for all four spaces to be even numbers, and from the pool of 6 numbers, you can take away the odd numbers 1, 3, and 5, leaving only three even numbers left. From here, you can do the same as part A but with 3’s instead: () or 34 to get **81**.

For part C, if the first, third, and fourth numbers are fixed, there is only 1 “free” position that can be any number, while there are 6 possible outcomes for the fixed positions since they have to be the same, which is the first number in the following expression: . In future cases, the exponent of the second number would be the number of freed positions there are.

The entirety of the 2nd number in the expression is to say, hey, the rest of the positions can be occupied by any of the 6 numbers, while the 1st is to represent those positions can only be those numbers.

The answer for this problem would be **36**.

**15.** A DNA sequence can be represented as the string of the letters ACTG (short for adenine, cytosine, guanine, and thymine).

1. How many DNA sequences are exactly 23 letters long?
2. Given a DNA sequence of length 23, how many single letter mutations are possible?
3. Given a DNA sequence of length 23, how many double letter mutations are possible?

In part A, it’s asking how many sequences are 23 letters long. This means there would be 23 spaces (and this will not be shown because that would be too long). Anyway, each space has 4 possibilities for ACTG. So, multiply them all together (423) to get **7.036874418E13** in scientific notation or **70368744177664**.

In part B, you have the total number of spaces, which is 23, choose 1. Remember in Mr. Cohen’s class, this is a combination problem. Combination is “*n* choose *r*”, or how many ways can you choose *r* items from a group of *n*. Since you are choosing 1 item from a group of 23, it would be () (please ignore the fraction line, it shouldn’t be there). This would equal .

Now, after that, you would multiply 3, because there are 3 possible mutations for that one space. In other words, say space A is going to mutate. There are three other letters that it could mutate to. In total, the equation would be .

The answer for this part would be **69**.

In part C, you do the same thing as part B, but instead, you’re choosing 2 items from a group of 23 because it is a double letter mutation, which is () (again, ignore the fraction line). This would equal .

Now, after that, instead of multiplying by 3, you would multiply by 9. Why? Double letter mutation. There are 2 spaces that could mutate, each with 3 possible mutations. If space A and G are going to mutate, each has 3 options to mutate to. Multiply them together () and you will get 9. In total, the equation would be .

The answer for this part would be **2277**.

**16.** How many different one-to-one functions can be defined that map the domain to the range ?

Let’s define what a one-to-one function is first. Basically, each input must have an output, and this output must not be shared. If there is a repetition of the output, then it is not.

To solve this problem, remember Mr. Cohen’s class. There are 14 options (outputs) to choose from, and since you want to match each input with an output, you get the term () or 14 outputs choose 7 inputs. This also works because you’re thinking of getting an output without repetition. This can also be expressed by . So, you would get **17297280** different functions, or **1.72973E7** in scientific notation.

**17.** How many different one-to-one functions can be defined that map the domain to the range such that *f* is NOT one-to-one?

You know that *f* is not one-to-one, which means there is a repetition of outputs. However, despite this, there are possibilities of having one-to-one functions.

Find the total number of functions first, which will be 157, which would equal 170859375.

Then, find the number of one-to-one functions and subtract. You can do this by recognizing one to one functions call for a combination (there is no need for order and repetition is not allowed) and that it needs the Product Principle. So, would be it. You would have 7 factors because there are 7 numbers in the domain. You would get 32432400.

The final answer of subtracting them both is **138426975** or **1.38427E8**.

**18.** How many different one-to-one functions can be defined that map the domain to the range ?

Refer to problem 16 for an elaborate explanation. Anyway, for this problem, there are 7 values in the domain and 7 values in the range. Since each input can be paired with an output (with no outputs leftover), you can do the factorial of 7 (7! or ) to get **5040** functions.

**19.** Count the number *X* of functions so that φ is not onto: .

According to the book, the symbol(*n*) is used to stand for the number of elements of Zn. In this case, it probably is a placeholder for *f*. Anyway, for a function to be onto, every target (range) needs to be hit. An onto function doesn’t have to be one-to-one. So, for a function not to be onto, at least one of the targets in the range should not be hit. From this point onward, let *f* represent the Greek symbol, let *S* represent the domain, and let *T* represent the range.

Think of the function as a collection. Think it in *n* element tuples (like writing outputs as {1,2,3} would mean f(1) = 1, and so forth, and this example would be a 3 element tuple). The *n* comes from the number of inputs in the domain.

Use the product principle. Product principle says to find out how many choices are allowed for each position. The same rule applies when counting functions.

The set has 5 values in the domain and 5 values in the range. So, there are 5 spaces:

To find how many aren’t onto, find how many possible functions can be defined from the set. Then, subtract the number of onto functions from the total.

Finding the total: There are 5 possibilities for each space, and there are 5 spaces, so using the product principle, you’d get 55, which would equal to 3125.

Finding out the number of onto functions: Do the same thing as problem 18. Since there are 5 spaces, you can just do 5! to get 120.

Subtract the two and you get **3005**.

**20.** Suppose that 671 tennis players want to play an elimination tournament. That means: they pair up, at random, for each round; if the number of players before the round begins is odd, one of them, chosen at random, sits out that round. The winners of each round, and the odd one who sat it out (if there was an odd one), play in the next round, till, finally, there is only one winner, the champion. What is the total number of matches to be played all together, in all the rounds of the tournament?

| **Equation**  *Players / 2 people per match* | **Answer**  *# of matches* | **Extra**  *Player that sits out* | **# of Players** | **Total # of Matches** |
| --- | --- | --- | --- | --- |
| 671 / 2 | 335 | 1 | 336 | 335 |
| 336 / 2 | 168 | 0 | 168 | 503 |
| 168 / 2 | 84 | 0 | 84 | 587 |
| 84 / 2 | 42 | 0 | 42 | 629 |
| 42 / 2 | 21 | 0 | 21 | 650 |
| 21 / 2 | 10 | 1 | 11 | 660 |
| 11 / 2 | 5 | 1 | 6 | 665 |
| 6 / 2 | 3 | 0 | 3 | 668 |
| 3 / 2 | 1 | 1 | 2 | 669 |
| 2 / 2 | 1 | 0 | 1 | **670 matches** |

This is basically an explanation of how it works. However, you can come to this conclusion through the Handshake method mentioned in Lesson 1.1.



In this instance, *n* would stand for the number of people in this case. So, *n* would be 671 people for this problem. You would also get the number **670** from this equation. Be sure to show your work and input the correct number, as well as adding units when not in WebWorK.