1. X is an integer when it is first declared and then it is given the value of 9.
   Then x changes itself by adding 10 making it 19. x is an variable holding an int

2. We believe that a variable is definitely a polymorphoc idea. This is because to
   be polymorphic, you need to be able to take multiple forms. A variable can be a
   number of things. You can have an int, like in problem 1, float, double, string
   and many more. This is what makes a variable polymorphic

3. A pointer is an object that stores not the value of which it is pointing to
   but rather the memory address of it. *p is storing the memory address of 9 and
   then altering that memory address to make it now 19

4. We think that pointers are not polymorphic. Since they are only storing the
   memory address, they are only doing one thing. They can point to other pointers
   and memory addresses can change, but we would not qualify this as a pointer
   being polymorphic

5. An array is a data structure that can store a fixed-size sequential collection
   of elementsof the same type. AN array can also be viewed as a collection of
   variables, and can be reached by using "array[]" and insert the name of the element
   inside of the brackets. Polymorphism in array is known as code that is able to
   change with differnt types of objects and behave differently with each

6. To be able to identify an element in an array, you would need to run through
   the array and land on the desired spot. You would need to go to array[i], where
   i is the location of the element that you want. So if it is the 5th element in
   the array, you would want to got to array[4]

7. A single array element is one specific element in the array. Similar to the answer to question 6, the single array element would be the element that you are tying to land on when you run through the array. If you have an array of length 10, and you want to get the element in the 10th spot, this is now your single array element

8. Some functions that we could preform on single array elements could be:
   *array_at so that we could fetch and store elements
   array_size so that we could find the size of the element
   array_get so that we could return a bit to see whether it is in the array
   array_put so that we could return the previous value of that element

9. A two dimensional array is similar to a regular array, but instead of there being one "column" in a 1D array, a 2D array has rows and columns. It is similar to that of a table used in mathematics

10. In order to identify an element in a 2D array, you will need to be able to find its location in the row that it is in, and its column. This is because there will be multiple elements in one row and multiple elements in one column. To find the single element array that you are looking for, you will need bot its "i" and "j" locations

11. In a two-dimensional array, we are locating an element at both its row and column that it lives in. For a one-dimensional array, we will only need the column that it lives in. Relating these elements would be having the element in the 1D array be the first element in the first column of any row. Meaning, the very far left of any row that the 1D array corresponds to. Everything else in that row would be if there were many 1D arrays. So the first column is the

first 1D array, 2nd column would be the 2nd 1D array etc. Having a 2D array eliminates the need to have multiple 1D arrays. The elements in a 2D array could be the element in a 1D array, but broken down to smaller amounts and seen more in depth. As an example, a 1D array could hold the size of ints being stored at certain places, and then a 2D array could hold the actual ints themselves