1) inches < 12
Initially we get 1 foot 15 inches, then we get 2 feet 3 inches, but if we go one further repetition (or iteration), we get 3 feet -9 inches, which is conventionally wrong because it is negative. No matter what example we use, we should stop converting it as soon as we have an inches value that is below 12 inches.

2) [2, 4, 6, 8, 10]
Each list element takes on the value of the element after it. For example, the first element takes on the value of the second element, and so on. Finally, i = 5 and so the fifth element of the list takes on the value 10, as stated after the loop.

3) [1, 3, 5, 4]
First, 4 is added to the end of the list. Then the value 1 is inserted into the second position, moving 3, 5, and 4 to the right. Finally, the value in the first position, namely 0, is removed from the list.

4) This would cause an error in the program and stop the program immediately (“crash”) before the program is finished. There is no list element with index (“position”) 5, since there are only 4 elements. It would not ignore your request, because it will try to do what it is told.

5) Choices A and D
If we take some sample grades and add 5 to them, if any grade is over 100 (say 102 or 103), then we still give them 100 according to the teacher’s intention. Therefore we are taking the minimum of 100 and the grade above 100 (like 102 or 103). Similarly, Choice D is correct because if a grade ends up higher than 100 after the 5 points are added, then we reset it to 100 rather than subtracting 5 points (as in Choice C), which would result in the same grade they had before the 5 points were added.

6) Both algorithms will work.
In the first algorithm, a single card is passed around and the sum updated by each student, meaning that the final value on the card is the sum of all students’ heights in class, which when divided by 32 results in the average height. In the second algorithm, students are paired with one another, but the sum of each pair is eventually added to the sum of another pair, and the resulting sum is added to the sum of two other pairs. By the end, you have the sum of all pairs’ heights in the class, which when divided by 32 results in the average height.

7) Choices 3 and 4
If a condition is true, then the user takes the path dictated by the “if” statement. If that condition is not true, then the user is forced onto the path dictated by the “else” statement. As you can see, the if/else block is being used to evaluate truth value as well as dictate path.

8) Choices 2 and 4
10 is the constant value 10 and cannot vary. “hello” is the word hello and cannot vary. On the other hand, the value of x can vary. For instance, it can take on the value 5 or 6 or 151. Similarly, because it is not in quotation marks, paintBrush does not represent a word or message. Try printing paintBrush in Python versus “paintBrush”. The former (paintBrush) will be undefined unless you give it a value beforehand. The latter (“paintBrush”) will literally print the word “paintBrush”, meaning it already has a value. Other variables include temp (from a previous problem), which took on a varying numerical value, and onTime, which took on a varying Boolean (True/False) value. Any word, whether a one-letter word like x or an 11-
letter word like *personality*, can be used as a variable if it is not in quotation marks, which would make it a "string", not a variable.

9) if item < x:
   x ← item

x is being used for a reason. And it has been set to 99 because that is a sufficiently high number where all values in *this* list are lower than it. Starting with the first item in the list, we compare the item to x, which results in x taking on the value of this item, namely 1. The next item is 0, which is less than x (which is 1), so x takes on the value 0. Since the next item, 4, is not less than the current x (which is 0), x remains at a value of 0. At this point, you should see that no remaining value in the list will be assigned to x unless it is less than 0, which none of the remaining numbers are.

To go further, if x was going to take on the first value in the list anyway (because all values in list are less than 99), than why not assign it the first value in the list to begin with? That is, set x = list[1] rather than to 99, so that this algorithm can be used on *any* list, not just lists whose values are lower than 99.

10) Choice 3
The original code shows that we are only talking about kids in the morningList to begin with. That means all we have to check is if they are also in the afternoonList, in which case they are in both lists, resulting in them being put onto the lunchList, as the blurb states. Choice 4 is wrong because it says you can be on either the morningList or the afternoonList but not necessarily both and you will still be put on the lunchList.

Furthermore, even if the “OR” is changed to “AND” in Choice 4, there is still excessive code because you have already checked if a child is in the morningList (from the original code), so why check again?