CSI 201
practice else if - learning to use while loops!

1. Goals for today: Practice, Explore, Practice!

2. Recall the syntax of an else-if statement from last time and let's practice!

```
double a = 0, b = 0; //declares and initializes
cin >> a >> b;
if(a > 0) {  //is a greater than b? (beware bad comments)
  //do stuff when ...


}
else if (a > b) {
  //do stuff when ...


}
else {
  //do stuff when ...


}
```

3. However, today we're going to focus on making things loop instead! It is often useful to be able to repeat a task. Sometimes this can be done to allow the user to repeat a process, to walk through every element in a list or to simply repeat a process until a certain condition is met. There are two major types of loops in C++, but we will focus only on one today: the while loop. Here is a sample while loop with syntax in place:

```
int i = 0;
while(i < 10) {
   cout << "The value of i is: " << i << endl;
   i++; //this means the same as i = i + 1; called: increment
}
cout << "The value of i after the loop is: " << i << endl;
```

4. Let's walk through the above code slowly and list its output!

5. What kinds of things do we notice? Can we tell what kind of value i might have after the loop without know what is inside the loop? Can we tell how many times the internal parts of the loop will run without knowing what is on the inside?

6. Let's take another example with some parts missing:

```
// A: stuff happens here
while(user_input < 0) {
   // B: some stuff happens in here
}
//C: other stuff here
```

7. Okay, what do we know about the variable user_input when we're at comment "C"

8. Can we tell how many times the loop runs with the information we currently have in hand?

9. What pieces of information do we need to be able to determine how many times the loop might run? Go back to our first while loop with the variable `i` above. What pieces of information do we need to know how many times that internal loop will run?

10. Okay, let's fill in the last while loop with some other sample code:

```
int user_input = 0;
cin >> user_input;
while(user_input < 0) {
  cerr << "User input was negative. Reading again." << endl;
  cin >> user_input;
}
cout << "Input must now be >= 0 (non-negative)" << endl;
```

11. These kinds of loops can be useful in "forcing" the user to give an appropriate value regardless of what your preferred range might be. Let's do another example:

```
int user_input = 0;
cin >> user_input;
while(user_input < 0 || user_input > 100) {
  cerr << "User input was not 0-100, reading again:" << endl;
  cin >> user_input;
}
cout << " WHAT GOES HERE? " << endl;
```

12. Here the two lines (||) mean OR. If either the input is less than 0 or if the input is more than 100 (or both) then we will enter the loop. So, what do we know about the input when we get to WHAT GOES HERE?

13. Let's try yet another:

```
int user_input = 0;
cin >> user_input;
while(user_input > 15 && user_input < 20) {
  cerr << "User input 16-19, reading again" << endl;
  cin >> user_input;
}
cout << " WHAT GOES HERE? " << endl;
```

14. In this case, the two ampersands (&&) mean AND. To enter this loop, the input must both be more than 15 and less than 20. So, what do we know about the input when we get to WHAT GOES HERE?

15. Loops are also VERY useful in repeating a task. Let's say we wanted to output 100 stars (*). We could write 100 cout statements or type the star 100 times in a single cout statement. Or, we can use a loop to repeat this task. Based on what we learned from the very first loop, there is a strategic way to do this, so let's take a look:

```
int i = 0;
while(i < 100) { //loop 100 times
  cout << "*"; // one star
  i++; //step up by 1 only
}
cout << endl; // end my line of stars
```

16. Time to do some practice! Try out the following scenarios.

17. Write a loop that outputs the numbers 1 to 1000.

18. Show how to output 30 dashes (minus signs) using a while loop.