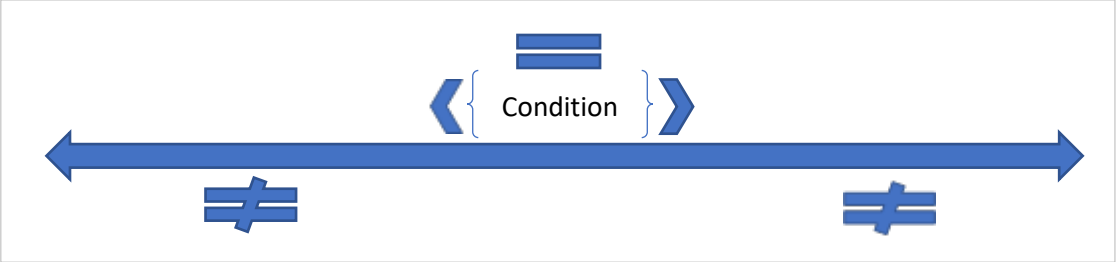
Intro to Programming

- Next Week is Midterm 1
- Today we'll start with the review quiz
- Today we're going to cover:
    - Review JS Conditionals
    - Demonstrate Nested Decisions
    - Learn the Switch/Case Statement
    - and learn about Iteration
- If we recall, last week we talked about conditional or decision branching
    - We discussed that if we look at a basic line, our conditions fall somewhere on that line



    - We used Boolean evaluation to establish the conditions
    - We discussed compound conditionals using "and" and "or"
    - We also discussed that decision making is a process of eliminate (either/or, not both)
    - Finally, we discussed how every decision flows to only one of two paths
        - In order to accommodate more paths, we primarily use two structures:
            - Nested Decisions, and the
            - Switch/Case Operator
- Nested Decisions:
    - We've already touched on the subject last week
    - Nesting is dependent on how precise or complex our decision is
    - Technically, there are two basic nesting structures:
        - if (condition) { if (condition) {…true…} else {…false…} } else {…false…}
          This structure, I prefer to consider as "embedded"
          While it is true that it is nested inside of another if, it does not build on that logic – it starts a whole new flow

        - if (condition) {…true…} else if (condition) {…true…} else {…false…}
          This structure is what we demonstrated in class last week
          This is more representative of complex logic
          An example similar to what we discussed is:

```
if (age < 60) {
        alert("Sorry, you are not eligible for retirement");
} else if (age < 65) {
        alert("Congrats! You are eligible for early retirement");
} else if (age === 65 && birthMonth === currentMonth) {
        alert("Almost there, you will be eligible to retire next month");
} else {
        alert("Congrats! You are eligible for your full retirement!");
}
```

- Switch/Case:
    - Nested Decisions can get complicated and hard to read (depending on formatting)
    - The other option is to use a Switch/Case statement instead of the if statement
    - The basic structure of a switch/case statement is as follows:
        - switch (expression) {

```
case x:
    // do something
    break;
case y:
    // do something
    break;
default:
    // do something
}
```

    - Switch is best suited for straight equality checks
    - For our previous example, most developers would say not to use a switch statement
    - As a rule of thumb, I use switch statement only if nesting exceeds 3 conditions
    - We can format the previous examples as such:
        - switch (true) {

```
case (age < 60):  // a normal case comparison is looking for equality
                  //  this statement will evaluate into either true or false
                  //  at that point the switch (true) is compare to the T/F
    alert('Sorry, you are not eligible for retirement');
    break;   //  we have to break out because switch falls through
case (age < 65):
    alert('Congrats! You are eligible for early retirement');
    break;
case (age === 65):
    if (birthMonth === currentMonth) {
        alert('Almost there, you will be eligible to retire next month');
    } else {
        alert('Congrats! You are eligible for your full retirement!');
    }
    break;
default:
    alert('invalid age entered');
}
```

        - view this code on CodePen (https://codepen.io/anon/pen/jodbZa?editors=1010 in case the link doesn't work for some reason)
- Now let's have a look at iteration:
    - Iteration is another fundamental building block of programming
        - We started with Variables
        - Proceeded to Conditionals
        - and now we're looking at Iteration (or looping)
    - Just as Variable alone were not enough, Conditionals alone are not enough – in order to truly make programming worth it, we need a mechanism to handle repetitive operations.

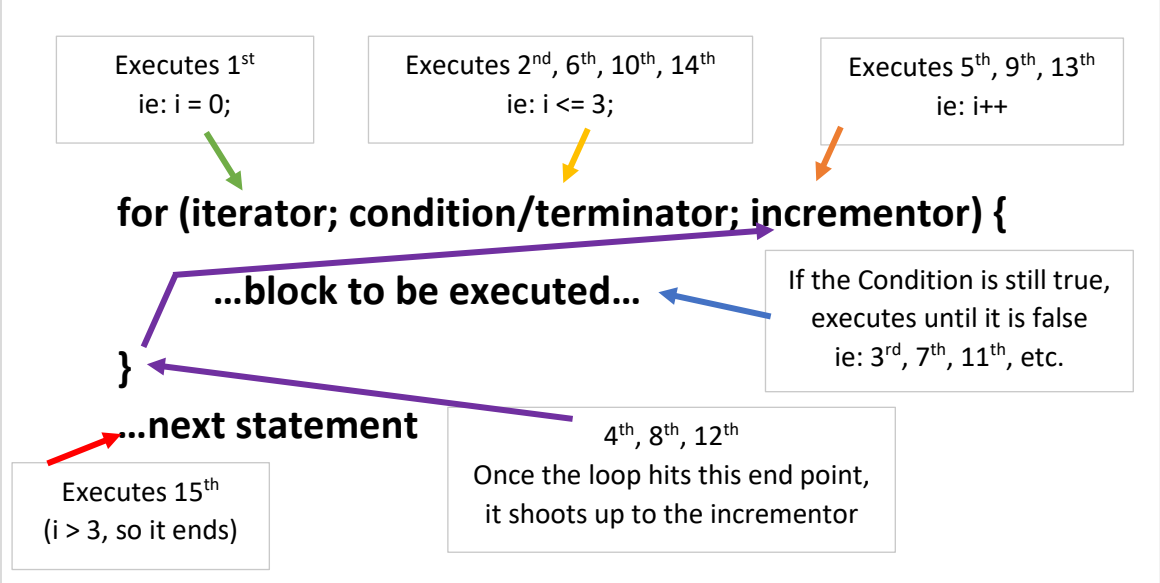
- For example – do we really want to code:
  - let count = 0;

```
count = count + 1; // 1
console.log(count);
count = count + 1; // 2
console.log(count);
count = count + 1; // 3
console.log(count);
count = count + 1; // 4
console.log(count);
count = count + 1; // 5
console.log(count);
count = count + 1; // 6
console.log(count);
count = count + 1; // 7
console.log(count);
count = count + 1; // 8
console.log(count);
count = count + 1; // 9
console.log(count);
count = count + 1; // 10
console.log(count);
```

  - a better solution would be: // this is the "for" statement

```
for (count = 1; count <= 10; count++) {
        console.log(count);
}
```

- The "for" statement is one of the predominately used looping structures in programming
  - The "for" loop is essentially a compound looping mechanism



- The “while” loop has essentially the same structure and flow except it’s all separate
  - let i = 0;
    while (i <= 3) {
    …block to be executed…
    i++;
    }
  - the main difference is that the iterator (or loop controller) is declared outside of the structure, but incremented inside of the structure
  - this structure lends itself to becoming an endless loop easier that the for loop
- There is a “do while” structure
  - The “while” and “for” loops check the condition before executing the block, where the “do while” will always execute the block at least once, and then check the condition
  - let i = 5;
    do {
    …block to be executed…
    i++;
    } while (i <= 3);
- There is also a “for in” loop that’s designed for iterating through objects, but it will either be discussed later in the semester, or next semester.