Whiteboard grading rubric for:
Date: Start time: End Time:

1. Interpreted the question
a. ___/2 points: Visually illustrated the problem domain
b. ___/2 points: Identified inputs and outputs
c. ___/2 points: Identified correct data structure

2. Solved the technical problem
a. ___/4 points: Solution works
b. ___/4 points: Code was syntactically correct
c. ___/4 points: Code was idiomatically correct
d. ___/4 points: Solution was the best possible option

3. Analyzed the proposed solution
a. ___/2 points: Stepped through their solution
b. ___/2 points: Big O time and space are considered

4. Communicated effectively throughout

a. ___/4 points:Verbalized their thought process
b. ___/2 points:Used correct terminology
c. ___/2 points:Used the time available effectively
d. ___/2Was not overconfident (not listening to suggestions)

e. ___/2 points:Was not underconfident (unsure of known algorithm)
f. ___/2 points:Whiteboard was readable (penmanship and spacing)

___/40 Total points

Giving up is an automatic fail, 80% required to pass

Problem 1
Given a calculator constructor

```
function calculator(){
   this.result = null;
}
```

1. Add a function to provide ‘add’ that takes a single input and adds to the result
2. Add a function to provide ‘multiply’ that takes a single input and multiplies the result
3. Clear function to clear result.
4. Stretch: make the functions chainable
5. HARD: Add a function to provide ‘factorial’ that takes no input but returns the factorial of the result 5! = 5 * 4 * 3 * 2 * 1 = 120 but doesn’t affect the result

Look for edge cases:
Deal with the fact that result is initialized to null - don’t try to add or multiply by null
Prevent divide by 0 if applicable
Allow for non integer results calculations - you can tell than about the parseFloat function
Don’t try to calculate a factorial on a non-integer result value.
Encourage the whiteboarder to ask questions and provide answers based on your knowledge of solutions.

```
calculator.prototype.add = function(val) {
  if (this.result === null) this.result = val;
  else parseFloat(this.result += val);
  return this; // this gives chaining
};

calculator.prototype.multiply = function(val) {
  if (this.result === null) this.result = val;
  else(parseFloat(this.result *= val));
  return this; // this gives chaining
};

calculator.prototype.clear = function() {
  this.result = null;
  return this;
};
Chained Calc:
var calc = new Calculator();
calc.add(3).multiply(1.2);


function factorialHelper(n) {
  var f = 1;
  for (var c = 1; c <= n; ++c) f *= c;
  return f;
}
calculator.prototype.factorial = function() {
  return factorialHelper(this.result);
};
```