Grade Claim 2 - The Sequel to the Critically Acclaimed Grade Claim One

Logan Hunt

13 November 2020

Contents

1 Intro: What grade do I claim?

A Bold Claim Deserves Enough Free, Good Grades. (ABCDEFGG)

```
d = (sum(list(map(lambda x: ord(x) - ord('A'), list("ABCDEFGG")))))
print(chr(d % 26))
```

2 Why do I claim this grade?

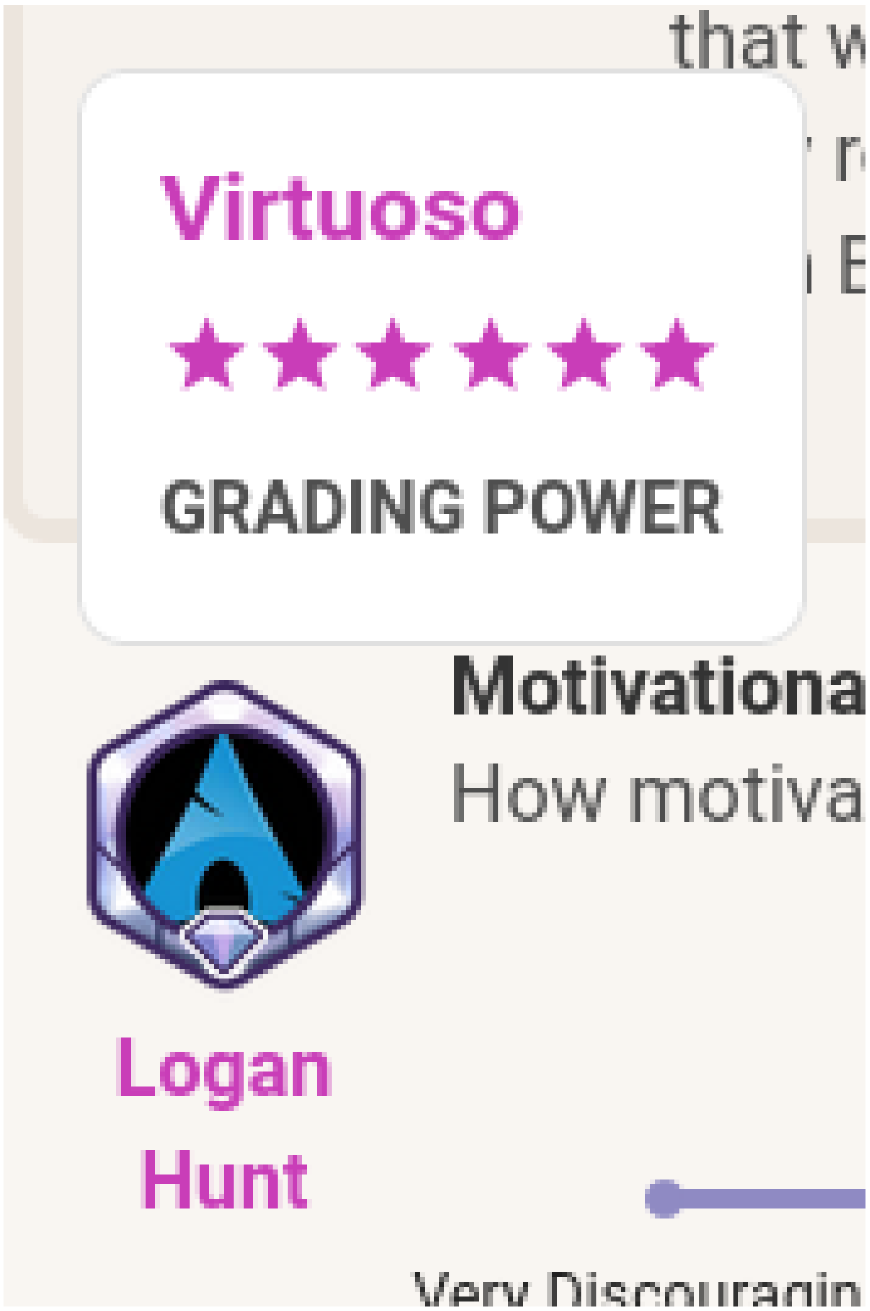
As mentioned in the grade-claim document, I believe I have gone above and beyond the requirements of "High Problem Solving, High Participation". In my mind, participation means how helpful one is to their team. In all of my activities on Kritik, I have provided supportive feedback that is unambigious. On Discord, I have helped others reach conclusions to their questions I can help them with, if I'm first anyways :).

3 Ok, do you have any evidence of these claims?

Of course I do! I will split the evidence into two portions: "High Participation" evidence, and "High Problem Solving" evidnece.

4 High Participation

Out of my evidence for my "High Participation", the simplest is my Kritik evaluation score, since it is unambiguous, and therefore finitely determinant. My Kritik profile has reached the highest Grading Power score possible:


that w
r
E
Virtuoso
★★★★★★
GRADING POWER
Motivationa
How motiva
Logan Hunt
Very Discouragin

Another piece of evidence for my "High Participation" is my conversation over Discord with one of my classmates, Aaron Picker. During this week's CDL we were asked to find the number of binary trees are possible with

4 nodes. After we found the number of trees are possible with 4 nodes, we decided to generalize the problem to n nodes. We worked together to implement a recursive solution that I came up with. I walked through the code with him for the example of 4 nodes, and explained the base cases as well as why the recursion works.

Here is the code we worked on:

```
def numTrees(n : int):
  if (n == 0 or n == 1):
    return 1
  total = 0
  for i in range(1,n+1):
    total += numTrees(n - i) * numTrees(i-1)
  return total

print(numTrees(4))
print(numTrees(3))
```

And for my last piece of evidence (though far from the actual last), I present my participation in the Office Hours I attended for Exercise 334.

5 High Problem Solving

For "High Problem Solving" evidence, I would invite the reader to take a look at these examples from my Ponder and Prove assignments.

1. Week 07

    - One simple example of my evidence that is present in Week 07 is in Exercise 319, where after I solve the problem, I play around with Bezout's theorem. The code is below:

      ```
      # Test bezout's theorem for ax + by = 1
      def testBezout(a, b, test_domain):
        solutions = []
        for x in test_domain:
          for y in test_domain:
            if a*x + b*y == 1:
              solutions.append((x, y))
        return solutions
      ```

```
# Test two co-primes 10 and 17
print(testBezout(10, 17, range(-100, 100)))

# Test two co-primes 5 and 4. A trivial solution would be (1, -1)
print(testBezout(5, 4, range(-10, 10)))
```

- Another example of a curious mind is in Exercise 321, where after I solve the problem, I experiment with the Congruence Mod M postulates that are presented in the book. Mainly, the addition postulate.
- My last example of going above and beyond was when I did Exercise 305. Instead of just finding the first CCS of length 10, I wrote a function to find the first CCS of any length. This proved to be very helpful, because the next exercise asked me to do just that, but I had already done it! Here is the code:

```
from helper import isPrime
def firstCCSLength(n, maxSearch):
  ccs = []
  for i in range(2, maxSearch):
    if len(ccs) == n:
      return ccs
    if isPrime(i):
      ccs = []
    else:
      ccs.append(i)

print(firstCCSLength(10, 10000))
```

2. Week 08

- Functional programming is definitely one of my weakenesses, but I aimed to strengthen it in Exercise 329, where I tried hard not to write a single for loop to practice.

```
from helper import get_lcm, findMMI
def solveSystem(mods, remainders):
  m = get_lcm(mods)
  o = list(map(lambda x: m // mods[x], range(len(mods))))
  y = list(map(lambda x: findMMI(o[x], mods[x]), range(len(mods))))

  roys = sum(list(map(lambda x: remainders[x] * o[x] * y[x], range(len(o)))))
  solution = roys % m

  return solution

print(solveSystem([7, 11, 13], [3, 5, 4]))
```

- In Exercise 334, I find a connection that was not **the** connection to find, but I was curious so I included it anyways. It's quite complex, and ends up probably being just a coincidence, but it is still interesting nonetheless. It is included in conjunction with the actual connection There is quite a bit of code, so I won't include it in this file, but it is available here: '`https://colab.research.google.com/drive/1EunGSvxmRruxJoLTpo8Gu9kPvZNUxCSc#scrollTo=AKyH_wAZ0dJ6`'
- In Exercise 335, I decide to play around with Object-Oriented design, and design my own ResidueNumberSystem class, though I have little experience with classes in Python. Below is some code to demonstrate this class:

```
from helper import ResidueNumberSystem, prod
a = ResidueNumberSystem([7,11,13])

def testRNSAddition():
  # Tests addition of two integers
  for i in range(1001):
    for j in range(1001):
      if not a.add(i, j) == a.intToRns(i + j):
        return False
  return True

def testRNSMultiplication():
  # Tests multiplication of two integers
  for i in range(1001):
    for j in range(1001):
      if not a.multiply(i, j) == a.intToRns(i * j):
        return False
  return True

print(testRNSMultiplication() == testRNSAddition() == True)
```

3. Week 09

    - I haven't quite finished the ponder and prove for this week quite yet as I'm writing this, but I do have at least one piece of evidence so far. Just like in my last example in Week 08, I wanted to practice even more with OOP, so I picked something that intrigued me: the permutation tree, and wrote a class for it. See code below for a demo of the tree doing a depth-first search with some pretty graphics describing the tree.

      ```
      from helper import PermutationTree

      b = PermutationTree(["A","B","C"])
      b.print_tree(b.root)
      ```

6 Woof, that was a lot of writing!

It sure was! I was going to write one more section about where I have taken this class' material outside of the assignments and participation, but in my opinion I have already provided enough concrete evidence for a strong claim. Plus I have to finish my work for my other high school classes. So until my next claim, I bid the reader farewell.