Matthew Beck and Spencer Hernandez **APMA 3100**
Spring 2022

Project 1: An Exploration of Counting Methods

On our honor, we have neither given or received aid on this assignment:

Matthew Timothy Beck

Spencer Lucas Henandez

For this project, you should not use any specialized functions or libraries (e.g. itertools) that your programming language might offer.

1. A student is registering for classes for next semester. The courses she can enroll in are calculus, physics, organic chemistry, history, literature, French, and astronomy. She has room for four classes in her schedule, at the following times: 9am, 10am, 11am, 12pm. She wants to count all the possible versions of her schedule. Additionally, the student's mother is helping her buy textbooks for her classes. Each course listed above has a specific textbook that is required with it. Therefore, a book order for the student will consist of four textbooks. Book orders are automatically listed in alphabetical order by the class subject (i.e. one book order could be astronomy, French, organic chemistry, physics). The student's mother wants to count all the possible book orders that she could be making.

   a. Make the beginnings of a carefully ordered list of all possible book orders. (Your list should include enough outcomes so that someone else can see and continue the pattern.)
   Astronomy, calculus, French, history, literature, organic chemistry, physics
   (astronomy, calculus, French, History), (astronomy, calculus, French, literature), (astronomy, calculus, French, organic chemistry), (astronomy, calculus, French, physics), (astronomy, calculus, history, literature), (astronomy, calculus, history, organic chemistry), (astronomy, calculus, history, physics), (astronomy, calculus, literature, organic chemistry), (astronomy, calculus, literature, physics), (astronomy, calculus, organic chemistry, physics), (astronomy, French, History, literature), (astronomy, French, History, organic chemistry), (astronomy, French, History, physics), (astronomy, French, literature, organic chemistry), (astronomy, French, literature, physics), (astronomy, French, organic chemistry, physics), (astronomy, History, literature, organic chemistry), (astronomy, History, literature, physics), (astronomy, History, organic chemistry, physics), (astronomy, literature, organic chemistry, physics),

b. Now make the beginnings of a carefully ordered list of all the possible versions of the student's *schedule,* assuming she can't register one class for multiple slots. (Your list should include enough outcomes so that someone else can see and continue the pattern.)

Astronomy, calculus, French, history, literature, organic chemistry, physics

- (astronomy, calculus, french, history)
- (astronomy, calculus, french, literature)
- (astronomy, calculus, french, organic chemistry)
- (astronomy, calculus, french, physics)
- (astronomy, calculus, history, french)
- (astronomy, calculus, history, literature)
- (astronomy, calculus, history, organic chemistry)
- (astronomy, calculus, history, physics)
- (astronomy, calculus, literature, french)
- (astronomy, calculus, literature, history)
- (astronomy, calculus, literature, organic chemistry)
- (astronomy, calculus, literature, physics)
- (astronomy, calculus, organic chemistry, french)
- (astronomy, calculus, organic chemistry, history)
- (astronomy, calculus, organic chemistry, literature)
- (astronomy, calculus, organic chemistry, physics)
- (astronomy, calculus, physics, french)
- (astronomy, calculus, physics, history)
- (astronomy, calculus, physics, literature)
- (astronomy, calculus, physics, organic chemistry)
- (astronomy, french, calculus, history)
- (astronomy, french, calculus, literature)
- (astronomy, french, calculus, organic chemistry)
- (astronomy, french, calculus, physics)
- (astronomy, french, history, calculus)
- (astronomy, french, history, french)
- (astronomy, french, history, literature)
- (astronomy, french, history, organic chemistry)
- (astronomy, french, history, physics)

c. Here is a program written in Python. Here is the same program written in a slightly better way that takes advantage of Python's ability to iterate through lists. Does this count the possible book orders, schedules, or neither? Briefly explain.
This program counts the number of schedules that the student could have. It uses 4 for-loops that will iterate through the array (possible classes) and will increase the number of orders by 1 for every time that the for loops are at different indexes. The program does not take into account the ordering of the classes.

d. Write a program using for loops that *lists* AND *counts* all the possible book orders. **Paste your code below. What are the first 20 outcomes that your program lists? What is the total number of outcomes?**

```
orders = 0
books = ["astronomy", "calculus", "french", "history", "literature", "organic
chemistry", "physics"]
N = len(books)

for i in range(N):
    for j in range(N):
        if i < j:
            for k in range(N):
                if k > i and k > j:
                    for l in range(N):
                        if l > i and l > j and l > k:
                            orders += 1
                            print("(" + books[i] + ", " + books[j] + ", " + books[k] + ", " +
books[l] + ")\n")
print(orders)
```

First 20 outcomes:

```
(astronomy, calculus, french, history)

(astronomy, calculus, french, literature)

(astronomy, calculus, french, organic chemistry)

(astronomy, calculus, french, physics)

(astronomy, calculus, history, literature)

(astronomy, calculus, history, organic chemistry)

(astronomy, calculus, history, physics)

(astronomy, calculus, literature, organic chemistry)

(astronomy, calculus, literature, physics)

(astronomy, calculus, organic chemistry, physics)

(astronomy, french, history, literature)
```

(astronomy, french, history, organic chemistry)

(astronomy, french, history, physics)

(astronomy, french, literature, organic chemistry)

(astronomy, french, literature, physics)

(astronomy, french, organic chemistry, physics)

(astronomy, history, literature, organic chemistry)

(astronomy, history, literature, physics)

(astronomy, history, organic chemistry, physics)

(astronomy, literature, organic chemistry, physics)

There are 35 total outcomes.

Note 1: Consider using something like < or <= instead of !=.
Note 2: The logic and code for this problem and in problems 2 & 3 can and should closely resemble the logic and code provided in problem 1c.

2. a. An 8-digit password is required to have three 0's and five 1's. You will determine how many unique passwords are possible.

First consider how you might notate possible outcomes in the process of listing them. For example, any of the following can represent the same outcome:

01011101
137 (the digits are the locations of the 0's)
24568 (the digits are the locations of the 1's)

Write a program that lists and counts the unique passwords. **Provide your code, the list of all outcomes, and the count. Briefly communicate which notation you are using for the outcomes.** (The list of outcomes you provide should match with the notation you're using in your code.) **Also briefly explain how your program works** (3-5 sentences) so that a programmer not familiar with this language could easily reproduce your results using a different language.

```
outcomes = 0
```

```
for i in range(1,9):
    for j in range(1,9):
        if i < j:
            for k in range(1,9):
                if k > i and k > j:
                    for l in range(1,9):
                        if l > i and l > j and l > k:
                            for h in range(1,9):
                                if h > i and h > j and h > k and h > l:
                                    print(str(i) + str(j) + str(k) + str(l) + str(h) + "\n")
                                    outcomes += 1
print(outcomes)
```

The notation that we decided to use is the index of where each 1 is in the password. The code is similar to the code in question 1, but here there are 5 for-loops in order to determine the index of each of the 5 1's. We have the 5 for-loops nested where each subsequent one only runs if the value of the current for loop is greater than the value of the for loops that are before it.

Outcomes:

```
12345

12346

12347

12348

12356

12357

12358

12367

12368

12378

12456
```

12457

12458

12467

12468

12478

12567

12568

12578

12678

13456

13457

13458

13467

13468

13478

13567

13568

13578

13678

14567

14568

14578

14678

15678

23456

23457

23458

23467

23468

23478

23567

23568

23578

23678

24567

24568

24578

24678

25678

34567

34568

34578

34678

35678

45678

There are 56 different passwords

Note: A working program will give you most of the credit for this section. However, you'll get full credit if you write your program in such a way that you never actually have all of the passwords stored in an array or list or any other container object simultaneously, but rather you simply create each one, count it, and then forget it.

b. Now suppose a 30-digit password is required to have thirteen 0's and seventeen 1's. You don't need to list all the possible passwords, but you need to determine how many exist. You may solve this however you wish (you don't have to write a program), but briefly compare your method to what you used in part (a) and justify your chosen method.
Using the formula to calculate how many different possible combinations there are given the fact that the sample space is 30 and the number of objects chosen in the set is 17 (or 13 depending on how one views the problem), I arrived at 119,759,850 different combinations. While there was no program written for this problem, the answer is still correct because you can arrive at the answer for part a by using the combination formula, 8 Choose 5, which yields 56.

3. An 8-digit password is required to have exactly three 0's. The other 5 digits can be any number 1-7, but numbers 1-7 may not be repeated.

> a. Again, consider different options for notating possible outcomes. (Hint: it may be easier to think of a way to notate the *location* of the 0's, rather than just writing the outcomes as a list of 8 digits) **Then make a list of at least 20 of the possible passwords by hand.** (List the outcomes in an ordered, intentional way so as not to miss any if you kept going.)
> 123
> 124
> 125
> 126
> 127
> 128
> 134
> 135
> 136
> 137
> 138
> 145

146
147
148
156
157
158
167
168

b. Write a program that lists and counts all the possible outcomes. Provide a pasted copy of your code, the first 100 passwords that it produces, and the final count.

```
outcomes = 0

for i in range(0,8):
  for j in range(0,8):
    if i < j:
      for k in range(0,8):
        if k > i and k > j:
          for l in range(0,8):
            if l > i and l > j and l > k:
              for h in range(0,8):
                if h > i and h > j and h > k and h > l:
                  for q in range(1,8):
                    for w in range(1,8):
                      if q != w:
                        for e in range(1,8):
                          if e != q and e != w:
                            for r in range(1,8):
                              if r != e and r != q and r != w:
                                for t in range(1,8):
                                  if t != e and t != r and t != w and t != q:
                                    password = [0,0,0,0,0,0,0,0]
                                    password[i] = q
                                    password[j] = w
                                    password[k] = e
                                    password[l] = r
                                    password[h] = t
                                    outcomes += 1
                                    if(outcomes <= 100):
```

```
                                                print(str(password[0]) +
str(password[1]) + str(password[2]) + str(password[3]) + str(password[4])  +
str(password[5]) + str(password[6]) + str(password[7]))

print(outcomes)
```

```
Outcomes:
12345000
12346000
12347000
12354000
12356000
12357000
12364000
12365000
12367000
12374000
12375000
12376000
12435000
12436000
12437000
12453000
12456000
12457000
12463000
12465000
12467000
12473000
12475000
12476000
12534000
12536000
12537000
12543000
12546000
12547000
12563000
12564000
12567000
12573000
12574000
```

12576000
12634000
12635000
12637000
12643000
12645000
12647000
12653000
12654000
12657000
12673000
12674000
12675000
12734000
12735000
12736000
12743000
12745000
12746000
12753000
12754000
12756000
12763000
12764000
12765000
13245000
13246000
13247000
13254000
13256000
13257000
13264000
13265000
13267000
13274000
13275000
13276000
13425000
13426000
13427000
13452000
13456000
13457000

```
13462000
13465000
13467000
13472000
13475000
13476000
13524000
13526000
13527000
13542000
13546000
13547000
13562000
13564000
13567000
13572000
13574000
13576000
13624000
13625000
13627000
13642000

Total Outcomes: 141120
```

Note 1: As in #2, you'll receive almost all of the credit here if your program works. But you'll get full credit if you write your program in such a way that you never actually have all of the passwords stored in an array or list or any other container object simultaneously, but rather you simply create each one, count it, and then forget it.
Note 2: Your program may take a long time to run if it displays all of the possible passwords to the screen. Remember that you only need the first 100 displayed.

c. Suppose you have a password with *m* digits required to have exactly *n* 0's. The other (m-n) digits can be any number 1-9 but numbers 1-9 may not be repeated. How many unique passwords exist? (Your answer should be a mathematical expression written in terms of *m* and *n*.)

Since there are m possible digits in the number and n of them are zeroes, we can use mCn (m choose n) to denote the number of possible outcomes. This number is inaccurate because it assumes that the possible numbers are either 0, or not zero, like in binary. To account for this, we have to consider all the different possible permutations that can result from 9 numbers in m-n different indices of the number, represented as 9 P (m-n). By multiplying 9 P (m-n) and m C n together, we get the following equation: $mCn \cdot 9P(m-n) = \frac{m!}{n!(m-n)!} \cdot \frac{9!}{(9-(m-n))!}$

4. A 5-letter password is to be constructed from the following letters: A, B, C, D, and E and the password must contain exactly 3 of the letters. (Obviously at least one of them will need to be used more than once to create a 5-letter password.)

You may solve problem 4 however you wish (with or without a program). If you write a program, you should clearly explain what each section of your code accomplishes. If you don't use a program, you should show all your work and justify each computation.

(a) Suppose A,B, and C are the letters selected and we use A twice, B twice, and C once. How many different passwords exist in this scenario?

In order to solve this problem we can make a smaller problem out of it. If we choose that C is in the first position only, then we know there remains 4 spots for the two As and two Bs to go into. We can use the equation of ${}_4C_2 = \frac{4!}{2!2!} = 6$. This means with C in the first position there are 6 different passwords that can be made. Moving C to the second, then the third and so on position we know that there will be 6 different passwords for each. So since C can be in 5 different positions, we get $5 * {}_4C_2 = 30$ different passwords.

(b) Suppose A,B, and C are the letters selected and there are no other restrictions. How many different passwords exist in this scenario?

We know from the previous scenario that there are 30 different outcomes when there is one letter with a frequency of 1 and 2 letters with frequency of 2. Because there are three of these outcomes (the scenarios where frequency of A=1, or B=1, or C=1), there are 90 outcomes. However, there are also cases where there are two letters with frequency 1 (ie: freq A and freq B is one but freq C is 3), to calculate the number of outcomes from this case, we can do 5*4 (number of positions for the first letter with freq 1 to go multiplied by number of positions for the second letter with freq 1 to go, the third letter is irrelevant). Because there are three instances of this case, the total number of outcomes for this case is 60. Therefore, because there are 90 outcomes of one case and 60 outcomes of another, there are 150 total outcomes.

(c) Now suppose we don't put any restrictions on the letters chosen. (It could be A,B, and C, or it could be A, C, and D, or it could be B, D, and E, etc.) How many different passwords exist in this scenario?

In b we determined that given 3 letters and the restrictions provided in this question there are 150 different passwords that you can create. So this is a question of how many different 3 letter combinations can we make with the given 5 letters. This is given by the equations ${}_5C_3 = \frac{5!}{3!2!} = 10$. So for each of these 10 different combinations we know that there are 150 different passwords so the total is $150 * {}_5C_3 = 150 * 10 = 1500$ different passwords.