Permutations and Factorials - Lab

Introduction

Before, we saw how the creation of a sample space is crucial in finding probabilities. The issue however is that, when the sample spaces grows bigger, it is not straightforward to manually compute the size of sample sets anymore.

Luckily, probability theory provides us with several formulas that can help us out. One set of formulas are known as **permutations**. This lab will help you get a better understanding of permutations, and provide practice!

Learning objectives

You will be able to:

- Understand how to count permutations, and how factorials are the building blocks of permutations
- Understand how to mathematically derive how many permutations there are for big sets
- Understand how to compute permutations of a subset
- Learn about permutations with replacement and repetition

A note on factorials

In the lecture, we talked about permutations in the context of the Michael Jackson coverband "Who's bad". We wanted to calculate how many ways we can order 3 songs in their setlist. We can use factorials for that, and how it's easy to see that you can use factorials for that. For 3 songs, this boils down to

In [1]:
```
setlist = 3*2*1
```

Now, writing this out is not an issue when n is small. What if n grows though? Imagine there are 10 songs in the setlist

In [2]:
```
setlist = 10*9*8*7*6*5*4*3*2*1
```

You wouldn't want to repeat this for 25 songs... Let's create a function for this!

What you'll do below is:

- create a function that takes one argument, n
- initialize prod as 1
- next, use prod in a while-loop (what is the stopping criterion?)
- update n so it decreases with value 1 each iteration. This way you essentially calculate $n * (n-1) * (n-2) * \ldots * (1)$

In [3]:
```
def factorial(n):
    prod = 1
    while n >= 1:
        prod = prod * n
        n = n - 1
    return prod
```

Now, test your function with n=20

In [4]:
```
factorial(20)
```

Out[4]:
```
2432902008176640000
```

Just so you know, Python has a built-in function `factorial` in the `math` library as well! Let's use our own function in this lab, but just use the `math` function once to check your previous answer!

In [5]:
```
import math

math.factorial(20)
```

Out[5]:
```
2432902008176640000
```

Some practice on permutations

Let's go back to the appointments exercise from the last lab. A teaching assistant is holding office hours so students can make appointments. She has 6 appointments scheduled today, 3 by male students, and 3 by female students. From what you learned in the permutations lecture, you now have a more structured way of getting to the whole sample space! Hint: a permutation with replacement is needed here. Think carefully of what needs to go in the denominator and the numerator respectively.

In [6]:
```
app_num = factorial(6)
app_num
```

Out[6]: 720

In [7]:
```
app_denom = factorial(3)*factorial(3)
print(app_denom)
```

```
36
```

In [8]:
```
app_total = app_num/app_denom
app_total
```

Out[8]: 20.0

Permutations: hack a phone

You misplaced your iPhone and are afraid it was stolen. Luckily, your iPhone needs a 4-digit code in order to get in. Imagine that a potential thief can do five attempts at getting the code right before the phone is permanently locked, how big is the chance the thief unlocks the phone?

Think about the sample space and the event space separately. What is the denominator for this problem?

In [9]:
```
denom_phone = 10**4
denom_phone
```

Out[9]: 10000

And the numerator?

In [10]:
```
numer_phone = 5
numer_phone
```

Out[10]: 5

In [11]:
```
prob_unlock = numer_phone/denom_phone
prob_unlock
```

Out[11]: 0.0005

Right before you lost your phone you ate a pretzel, and you are pretty sure a grease pattern was left on the four crucial digits of your screen. The four letters in your access code are 3,4,7 and 8, and you realize that this information can increase the thief's chances massively. Assuming the thief interprets the smudgemarks in an intelligent way, what are the chances that the phone will be unlocked successfully?

In [12]:
```
denom_phone_smudge = factorial(4) #or math.factorial(4)
denom_phone_smudge
```

Out[12]: 24

In [13]:
```
numer_phone_smudge = 5
numer_phone_smudge
```

Out[13]: 5

In [14]:
```
prob_unlock_smudge = numer_phone_smudge/denom_phone_smudge
prob_unlock_smudge
```

Out[14]: 0.20833333333333334

Now, imagine you chose an iphone access code containing 3 different numbers, with numbers 2,7 and 8 (the code is still 4 digits). Now, the thief knows 1 number was reused, but he doesn't know which one. what is the probability now that the phone will be unlocked successfully?

In [15]:
```
denom_phone_smudge_2 = ((4*3*2*1)/2) * 3 #or use math.factorial(4)
denom_phone_smudge_2
```

Out[15]: 36.0

In [16]:
```
numer_phone_smudge_2 = 5
numer_phone_smudge_2
```

Out[16]: 5

In [17]:
```
prob_unlock_smudge_2 = numer_phone_smudge_2/denom_phone_smudge_2
prob_unlock_smudge_2
```

Out[17]: 0.1388888888888889

Permutations to find the sample and event space

What are the odds of throwing a "full house" when throwing 5 dices? Recall, a full house means that you'd throw a three of a certain number along with a pair of a different number.

a) sample space

First, calculate the sample space. recall that replacement is possible here.

In [18]:
```
sample_space_fh = 6**5
sample_space_fh
```

Out[18]: 7776

b) event space

Next, calculate the event space. The best way to think of the event space here, is split it up in 2 parts:

- first, try to constrain your problem to a more specific problem, let's say, how many ways can we throw a full house if we have a pair of 4s and three 6s?
- next, extend your problem asking yourself how many *different* full houses are possible.
- multiply the two!

In [19]:
```
ways_to_throw_given_fh= factorial(5)/ (factorial(3)* factorial(2)) # permutation with repetitions
ways_to_throw_given_fh
```

Out[19]: 10.0

In [20]:
```
diff_fhses = math.factorial(6)/math.factorial(4)
diff_fhses
```

Out[20]: 30.0

Then the event space is

In [21]:
```
event_space_fh =  ways_to_throw_given_fh * diff_fhses
event_space_fh
```

Out[21]: 300.0

c) Probability of full house

In [22]:
```
prob_fh = event_space_fh/sample_space_fh

prob_fh
```

Out[22]: 0.038580246913580245

Summary

Great job! You got quite some practice in on permutations and factorials, and were even able to use it to calculate probability. Now we'll move over to another concept in combinatorics: combinations.