#Importing Library: Numpy

(From Last week's Lab, you can find more information about \*\*"NumPy"\*\* library).

NumPy is the fundamental Python library for numerical computing.

import numpy as np

#Importing Library: SciPy

(From Last week's Lab, you can find more information about \*\*"SciPy"\*\* library).

\*\*SciPy\*\* stands for Scientific python. For some statistical calculation we do need Scipy e.g, during last week, you have used stats from Scipy to calculate "Mode".

from scipy import stats

#Importing Library: Math

\*\*Math\*\* is a library you can apply for performing mathematical tasks by using different Math methods in this library.

import math as mt

#Importing Library: Random

Python has a built-in library \*\*random\*\* which can be used for generating random numbers/values

import random as rnd

#Probability

We first see how a sample space is defined in Python using the example of a coin. A sample space is the set of all the possible outcomes an action could produce. Let’s take the simple action of flipping a coin. The coin will land on either heads or tails. Thus, the coin flip will produce one of two measurable outcomes: heads or tails. By storing these outcomes in a Python set, we can create a sample space of coin flips.

sample\_space = {'Heads', 'Tails'}

Suppose we choose an element of sample\_space at random. What fraction of the time will the chosen element equal Heads? Well, our sample space holds two possible elements. Each element occupies an equal fraction of the space within the set. Therefore, we expect Heads to be selected with a frequency of 1/2. That frequency is formally defined as the probability of an outcome. All outcomes within sample\_space share an identical probability, which is equal to 1 / len(sample\_space).

Example code is given below

sample\_space = {'Heads', 'Tails'}

probability\_heads = 1 / len(sample\_space)

print(f'Probability of choosing heads is {probability\_heads}')

# Permutations of a Python string

If we are given a Python string and asked to find out all the ways its letters can be arranged, then the task can easily be achieved by the permutations() function

import itertools

st = "ABC"

per = itertools.permutations(st)

for val in per:

print(\*val)

#Question 1

Using the code given above calculate the number of permutations for the string "CHOCOLATE"

# Permutations of multiple numbers

The permuatations() function takes an iterable argument, therefore in order to find out permutations of numbers, we need to pass the numbers as a list, set, or tuple.

import itertools

values = [1, 2, 3]

per = itertools.permutations(values)

for val in per:

print(\*val)

# Permutations with certain number of elements

Similar, to the concept of ‘nPr’, which states “Arranging r elements out of n”, this can be achieved by passing an integer after the set of elements.

import itertools

values = [1, 2, 3, 4]

per = itertools.permutations(values, 2)

for val in per:

print(\*val)

# Finding Combinations

The term Combinations, refer to the ways of picking up elements from a set of objects. The itertools library provides a method combinations() for exactly this functionality.

One thing to note here is that, picking a set of objects does not involve arranging. The combinations() function takes two arguments:

1. The set of values

2. An integer denoting the number of values to be picked for a combination.

Combinations for letters in a word

Given a word, if we need to find all combinations containing exactly 2 letters from the word, then combinations() is the go-to function.

import itertools

st = "ABCDE"

com = itertools.combinations(st, 2)

for val in com:

print(\*val)

# Combinations of set of numbers

Similar to the combinations result we got for letters in a word, it can be achieved for numbers in a list.

import itertools

values = [1, 2, 3, 4]

com = itertools.combinations(values, 2)

for val in com:

print(\*val)

# Combinations for repeated numbers

To further explain the above Note, let us run an example for it.

import numpy as np

HP\_Var = np.loadtxt("./HP.csv", dtype=int)

print (HP\_Var)

Mean\_of\_HP= np.mean(HP\_Var)

print (Mean\_of\_HP)

import itertools

values = [1, 1, 2, 2]

com = itertools.combinations(values, 2)

for val in com:

print(\*val)

# Combinations of numbers with itself

There is yet another function related to permutations and combinations in the itertools library called combinations\_with\_replacement(). This function is a variation of combinations() function, with a slight difference that it includes combinations of elements with themselves.

import itertools

values = [1, 2, 3, 4]

com = itertools.combinations\_with\_replacement(values, 2)

for val in com:

print(\*val)

# Factorial

Factorial is the most important concept in Permutations and Combinations.

Here’s a simple recursive function which will help in calculating the factorial

def factorial(n):

if n==1: return 1

else: return n \* factorial(n-1)

# Permutations

You have a website on which users can register. They need to provide a password that needs to be exactly 8 characters long, and characters cannot repeat. We first need to determine how many characters and digits there are in the English alphabet:

the number of letters: 26

the number of digits: 10

Which is 36 in total. So n = 36. r would then be 8, because the password needs to be 8 characters long.

Example code given below

def factorial(n):

if n==1 : return 1

else: return n \* factorial(n - 1)

def permutation\_without\_repetition(n,r):

return (factorial(n)/(factorial(n - r)))

print(permutation\_without\_repetition(36,8))

#Question 2

Using the example code given above, solve the following question by writing a code, In how many different ways can the letters of the word 'OPTICAL' be arranged so that the vowels always come together?

# Question 3

Using the hints given in the codes above showing how to calculate permutations, write a code to calculate the possible number of combinations, how many ways can you choose 5 people from a group of 10 for a football team?

#Calculating Probability

Here, we will see examples of probability calculation from a random variable. To do so, we will use \*\*random\*\* library from python.

From a tossing coin we have two events in a sample space. The following code calculates the probability of getting head when we do hundred trials of flipp

def toss\_coin(n):

trials = []

for i in range(n):

trials.append(rnd.random() <= 0.5) #we are considering 50% or below will be the probability of getting heads from random flips

return sum(trials) #returns the event count for observed heads from coin flips

def probability(n, e):

probability\_heads=(e/n)

return probability\_heads

n=10

probability\_heads =probability(n, toss\_coin(n))

print(probability\_heads)

n=100

probability\_heads =probability(n, toss\_coin(n))

print(probability\_heads)

n=1000

probability\_heads =probability(n, toss\_coin(n))

print(probability\_heads)

#Question: 4

Now, calculate the probability of finding "Tails" when you toss a fair coin.