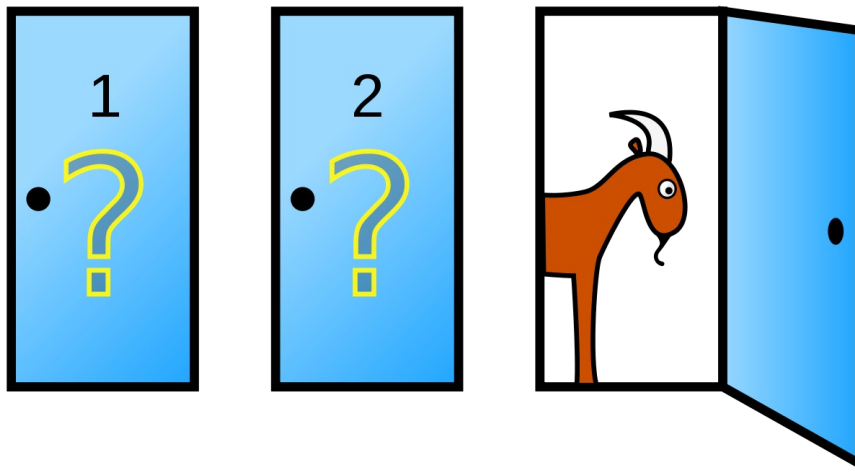


## El famoso Problema del Laberinto

Este problema aparece en la película '21 Blackjack' y en una charla que dio Adrian Paenza en un evento de SAS Argentina también lo comentó y generó mucha discusión entre coworkers.


'Hay tres puertas cerradas, detrás de una de ellas hay un premio valioso, y detrás de las otras una cabra (no sé por qué una cabra, pero al parecer están devaluadas), vos elegís una puerta y luego de elegirla el moderador que sabe que hay detrás de cada puerta, abre una de las que no elegiste que tiene una cabra, y te ofrece la posibilidad de elegir quedarte con tu elección o elegir la otra que quedó cerrada'.



1. A continuación, puede seguir una de las dos siguientes estrategias:
  - o Quedarse con la puerta seleccionada al azar.
  - o Seleccione la otra puerta que queda, de contenido desconocido.

Calcula las probabilidades de obtener el premio con cada una de las estrategias.

## Integración de Monte Carlo en Python



La integración de Monte Carlo es un proceso de resolución de integrales que tienen numerosos valores para integrar. El proceso de Monte Carlo utiliza la teoría de los grandes números y el muestreo aleatorio para aproximar valores que están muy cerca de la solución real de la integral. Funciona sobre la media de una función denotada por  $\langle f(x) \rangle$ . Luego podemos desarrollar  $\langle f(x) \rangle$  como la suma de los valores dividida por el número de puntos en la integración y resolver el lado izquierdo de la ecuación para aproximar el valor de la integración en el lado derecho. La derivación es la siguiente.

$$\begin{aligned}\langle f(x) \rangle &= \frac{1}{b-a} \int_a^b f(x) dx \\ (b-a) \langle f(x) \rangle &= \int_a^b f(x) dx \\ (b-a) \frac{1}{N} \sum_{i=1}^N f(x_i) &\approx \int_a^b f(x) dx\end{aligned}$$

Donde N = número de términos utilizados para la aproximación de los valores.

Ahora primero calcularíamos la integral usando el método de Monte Carlo numéricamente y luego finalmente visualizaríamos el resultado usando un histograma utilizando la biblioteca de Python matplotlib.

Módulos utilizados:

Los módulos que se utilizarían son:

- **Scipy** : se utiliza para obtener valores aleatorios entre los límites de la integral. Se puede instalar mediante:

- ```
pip install scipy # para windows
```
- o  
Instalación de pip3 scipy # para linux y macos

- 
- **Numpy** : se utiliza para formar matrices y almacenar distintos valores. Se puede instalar mediante:

- ```
pip install numpy # para windows
```
- o

Instalación de pip3 numpy # para linux y macos

- [Matplotlib](#) : se utiliza para visualizar el histograma y el resultado de la integración de Monte Carlo.

- 

pip install matplotlib # para windows

o

Instalación de pip3 matplotlib # para linux y macos

*Ejemplo 1:*

La integral que vamos a evaluar es:

$$\int_0^{\pi} \sin x \, dx$$

Una vez que terminamos de instalar los módulos, ahora podemos comenzar a escribir código importando los módulos necesarios, scipy para crear valores aleatorios en un rango y el módulo NumPy para crear matrices estáticas, ya que las listas predeterminadas en Python son relativamente lentas debido a la asignación de memoria dinámica.

Luego definimos los límites de integración que son 0 y pi (usamos np.pi para obtener el valor de pi).

Luego creamos una matriz numpy estática de ceros de tamaño N usando np.zeros(N). Ahora iteramos sobre cada índice de la matriz N y lo llenamos con valores aleatorios entre a y b.

Luego creamos una variable para almacenar la suma de las funciones de diferentes valores de la variable integral. Ahora creamos una función para calcular el seno de un valor particular de x.

Luego iteramos y sumamos todos los valores devueltos por la función de x para diferentes valores de x a la variable integral.

Luego usamos la fórmula derivada anteriormente para obtener los resultados. Finalmente, imprimimos los resultados en la línea final.

```

# importing the modules
from scipy import random
import numpy as np

# limits of integration
a = 0
b = np.pi # gets the value of pi
N = 1000

# array of zeros of length N
ar = np.zeros(N)

# iterating over each Value of ar and filling
# it with a random value between the limits a
# and b
for i in range (len(ar)):
    ar[i] = random.uniform(a,b)

# variable to store sum of the functions of
# different values of x
integral = 0.0

# function to calculate the sin of a particular
# value of x
def f(x):
    return np.sin(x)

# iterates and sums up values of different functions
# of x
for i in ar:
    integral += f(i)

# we get the answer by the formula derived above
ans = (b-a)/float(N)*integral

# prints the solution
print ("The value calculated by monte carlo integration is
{}.".format(ans))

```

*El valor calculado por la integración de Monte Carlo es 2.0256756150767035.*

El valor obtenido es muy cercano a la respuesta real de la integral que es 2.0.

Ahora bien, si queremos visualizar la integración mediante un histograma, podemos hacerlo utilizando la biblioteca matplotlib. Nuevamente importamos los módulos, definimos los límites de integración y escribimos la función seno para calcular el valor seno para un valor particular de  $x$ . A continuación, tomamos una matriz que tiene variables que representan cada haz del histograma. Luego, iteramos a través de  $N$  valores y repetimos el mismo proceso de crear una matriz de ceros, llenándola con valores  $x$  aleatorios, creando una variable integral que sume todos los valores de la función y obteniendo la respuesta  $N$  veces, cada respuesta representando un haz del histograma. El código es el siguiente:

```
#importing the modules
from scipy import random
import numpy as np
import matplotlib.pyplot as plt

# limits of integration
a = 0
b = np.pi # gets the value of pi
N = 1000

# function to calculate the sin of a particular
# value of x
def f(x):
    return np.sin(x)

# list to store all the values for plotting
plt_vals = []

# we iterate through all the values to generate
# multiple results and show whose intensity is
# the most.
for i in range(N):

    #array of zeros of length N
    ar = np.zeros(N)

    # iterating over each Value of ar and filling it
    # with a random value between the limits a and b
```

```

for i in range (len(ar)):
    ar[i] = random.uniform(a,b)

# variable to store sum of the functions of different
# values of x
integral = 0.0

# iterates and sums up values of different functions
# of x
for i in ar:
    integral += f(i)

# we get the answer by the formula derived adobe
ans = (b-a)/float(N)*integral

# appends the solution to a list for plotting the graph
plt_vals.append(ans)

# details of the plot to be generated
# sets the title of the plot
plt.title("Distributions of areas calculated")

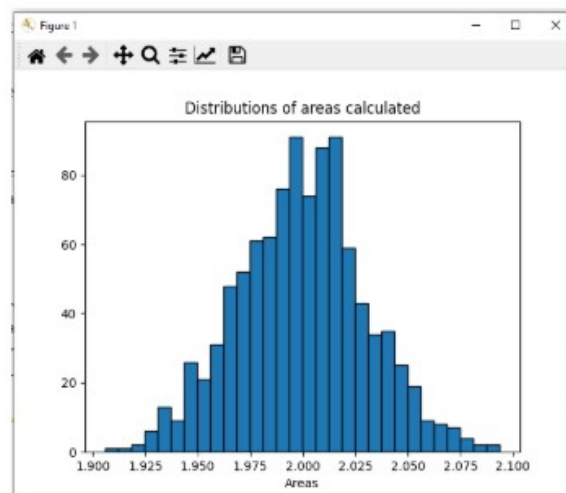
# 3 parameters (array on which histogram needs
plt.hist (plt_vals, bins=30, ec="black")

# to be made, bins, separators colour between the
# beams)
# sets the label of the x-axis of the plot
plt.xlabel("Areas")
plt.show() # shows the plot

```

Aquí como podemos ver, los resultados más probables según este gráfico serían 2.023 o 2.024, lo que nuevamente está bastante cerca de la solución real de esta integral que es 2.0.

**Producción:**



*Ejemplo 2:*

La integral que vamos a evaluar es:

$$\int_0^1 x^2 dx$$