Actividad 1

Nombre: Nuria Arroyo Bustamante

Matrícula: 173605

Triángulo de Pascal

Ejercicio: ¿Puedes probar que la Recursión sea más tardada que los loops? Usa el ejemplo del triángulo de Pascal y adapta tu código para que mida tiempos de ejecución. Para observar diferencias notables, calcula la n-ésima fila, para una n lo suficientemente grande.

Agrega una explicación en texto y carga en formato pdf tu respuesta en Blackboard.

En el primer bloque de código se replica el código visto en clase del algoritmo para obtener la k-ésima fila del triángulo de Pascal a través de la recursividad. En este método se obtiene el factorial a partir de una función recursiva. Esta función sirve como base para poder realizar el experimento de comparación.

In [1]:
```
#recursividad

def factorial_rec(num):
    if num > 0:
        return int(num*factorial_rec(num-1))
    else:
        return 1

def combinatoria_rec(num1, num2):
    return int(factorial_rec(num1) / (factorial_rec(num2)*factorial_rec(num1-num2)))


def pascal_combinatoria_rec(num):

  row=[]
  for j in range (0, num+1):
    row.append( combinatoria_rec(num, j) )
  return row
```

In [2]:
```
k = 10
n = 4
print("Pascal recursivo, comprobar los resultados invariantes de las funciones")
pascal_combinatoria_rec(k)
print( "La la k-ésima fila del triángulo de Pascal, con k=",k," es:", pascal_combinatoria_rec(k) )
factorial_rec(n)
print("El factorial de n=",n," es:", factorial_rec(n) )
combinatoria_rec(k,n)
print("La combinatoria de k=",k," y n=",n," es:", combinatoria_rec(k,n) )
```

```
Pascal recursivo, comprobar los resultados invariantes de las funciones
La la k-ésima fila del triángulo de Pascal, con k= 10  es: [1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1]
El factorial de n= 4  es: 24
La combinatoria de k= 10  y n= 4  es: 210
```

En este segundo bloque de código, en lugar de utilizar una función recursiva para calcular el factorial, se implementa una función basada en un loop. Esta función iterativa permite calcular el factorial de manera más eficiente y sirve como base para el experimento de comparación de tiempos de ejecución entre ambos enfoques.

In [3]:
```
# Loops

#recursividad

def factorial_loop(num):
    if num > 0:
        factorial = 1
        for i in range(num-1):
            factorial = factorial*(num - 1)
        return factorial
    else:
        return 1

def combinatoria_loop(num1, num2):
    return int(factorial_loop(num1) / (factorial_loop(num2)*factorial_loop(num1-num2)))


def pascal_combinatoria_loop(num):

  row=[]
  for j in range (0, num+1):
    row.append( combinatoria_loop(num, j) )
  return row
```

In [4]:
```
k = 10
n = 4
print("Pascal loop, comprobar los resultados invariantes de las funciones")
pascal_combinatoria_loop(k)
print( "La la k-ésima fila del triángulo de Pascal, con k=",k," es:", pascal_combinatoria_loop(k) )
factorial_loop(n)
print("El factorial de n=",n," es:", factorial_loop(n) )
combinatoria_loop(k,n)
print("La combinatoria de k=",k," y n=",n," es:", combinatoria_loop(k,n) )
```

```
Pascal loop, comprobar los resultados invariantes de las funciones
La la k-ésima fila del triángulo de Pascal, con k= 10  es: [1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1]
El factorial de n= 4  es: 24
La combinatoria de k= 10  y n= 4  es: 210
```

In [5]:
```
# experimento
import time

def experimento(sims, k):
    recursivas = []
    loops = []

    for i in range(sims):
        start = time.time()
        factorial_rec(k)
        end = time.time()
        recursivas.append(end - start)

        start = time.time()
        factorial_loop(k)
        end = time.time()
        loops.append(end - start)

    return recursivas, loops
```

In [6]:
```
recursivas, loops = experimento(3,10)
```

In [7]:
```
import numpy as np
import pandas as pd

import plotly.graph_objects as go
import scipy.stats as stats

#grafico de resutados

def mostrar_resultados(vec1, vec2, label1='Recursivo', label2='Loop'):
    # Estadísticos descriptivos
    stats_dict = {}
    for vec, label in zip([vec1, vec2], [label1, label2]):
        stats_dict[label] = {
            'media': np.mean(vec),
            'mediana': np.median(vec),
            'moda': stats.mode(vec, keepdims=True)[0][0],
            'varianza': np.var(vec),
            'asimetria': stats.skew(vec),
            'kurtosis': stats.kurtosis(vec)
        }

    # DataFrame para mostrar estadísticos
    df_stats = pd.DataFrame(stats_dict)

    # Histogramas side by side
    fig = go.Figure()
    fig.add_trace(go.Histogram(x=vec1, name=label1, marker_color='blue', opacity=0.7))
    fig.add_trace(go.Histogram(x=vec2, name=label2, marker_color='red', opacity=0.7))
    fig.update_layout(barmode='group')

    fig.update_layout(title='Histogramas y KDE de tiempos de ejecución', barmode='overlay')

    # Boxplots
    fig_box = go.Figure()
    fig_box.add_trace(go.Box(y=vec1, name=label1, boxmean=True))
    fig_box.add_trace(go.Box(y=vec2, name=label2, boxmean=True))
    fig_box.update_layout(title='Boxplot de tiempos de ejecución')

    # Pruebas de normalidad
    shapiro1 = stats.shapiro(vec1)
    shapiro2 = stats.shapiro(vec2)
    normal1 = shapiro1.pvalue > 0.05
    normal2 = shapiro2.pvalue > 0.05

    print("Estadísticos descriptivos:\n", df_stats)
    print(f"\nPrueba Shapiro-Wilk {label1}: p={shapiro1.pvalue:.4f} {'Normal' if normal1 else 'No normal'}")
    print(f"Prueba Shapiro-Wilk {label2}: p={shapiro2.pvalue:.4f} {'Normal' if normal2 else 'No normal'}")

    # Prueba de hipótesis
    if normal1 and normal2:
        ks = stats.ks_2samp(vec1, vec2)
        print(f"\nKolmogorov-Smirnov: p={ks.pvalue:.4f} {'Distribuciones similares' if ks.pvalue > 0.05 else 'Distribuciones diferentes'}")
    else:
        mw = stats.mannwhitneyu(vec1, vec2)
        print(f"\nMann-Whitney U: p={mw.pvalue:.4f} {'Distribuciones similares' if mw.pvalue > 0.05 else 'Distribuciones diferentes'}")

    fig.show()
    fig_box.show()
```

In [9]:
```
# ejecuciond del experimento
sims = 1000
k = 5
recursivas, loops = experimento(sims,k)
mostrar_resultados(recursivas, loops, label1='Recursivo', label2='Loop')
```

```
Estadísticos descriptivos:
                Recursivo          Loop
media       2.841949e-07  1.964569e-07
mediana     2.384186e-07  2.384186e-07
moda        2.384186e-07  2.384186e-07
varianza    1.143198e-13  2.529669e-14
asimetria   2.392936e+01  1.571108e+00
kurtosis    6.830368e+02  9.909568e+00

Prueba Shapiro-Wilk Recursivo: p=0.0000 No normal
Prueba Shapiro-Wilk Loop: p=0.0000 No normal

Mann-Whitney U: p=0.0000 Distribuciones diferentes
```

La media de los experimetos de loops es menos a la media de los experimentos de recursión. La meediana y moda de ambos experimentos son iguales. Al observar las varianzas vemos que la varianza de los experimentos de loops es menor a la varianza de los experimentos de recursión. De ehcho esta es 4.5 veces mas grande.

En el caso de la recursion se puede observar que la distribucion tiene asimatria y kurtosis que indican un cola pesada a la derecha. Esto se interprea con varios valores con latencia mayor. En el caso de los loops la asimetria es casi nula y la kurtosis inican una distrivucion co colas pero no tan pesadas.

Se pide normalidad para ver si es prudente utlizar la prueba de kolmogorov-smirnov. Sin embargo, en ambos casos se rechaza la hipotesis nula de normalidad. Es por esto que se utiliza la prueba de mann-whitney para comparar ambas distribuciones. En este caso se rechaza la hipotesis nula de que ambas distribuciones son iguales. Podemos concluir que son diferentes. Y por los estadicos descriptivos podemos concluir que la distribucion de los loops es menor a la de recursión.

Se puede observar quela función basada en loops es significativamente más rápida que la función recursiva. Esto se debe a que la recursión implica múltiples llamadas a funciones y puede llevar a una sobrecarga considerable, mientras que los loops son generalmente más eficientes en términos de tiempo de ejecución y uso de memoria.