TALLLER N°1

- Se debe presentar:
 - Notebook
 - Enlace al repositorio es: https://github.com/Nattyrd/Metodos-Numericos-2025B

KEVIN GARCIA 2025B

Ejercicio

Escribir la ecuación Navier Stokes.

Ecuacion de Navier-Stokes

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u}\right) = -\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{f}$$

Ejercicio 1

La sumatoria 1+1/2+1/4+1/8... tal que el error absoluto $e_{abs}<10^{-1}.$

```
In [2]: import random
   import time
   import math
   import matplotlib.pyplot as plt
```

```
In [3]: def serie_error_abs(t):
    limite = t
    suma = 0.0

    for x in range(limite + 1): # range no incluye el valor final
        suma += 1 / (2 ** x)

    print("La suma de la serie hasta 1/2^", limite, "es:", suma)

# Calcular el error absoluto
    error_abs = abs((suma - 2))
    print("El error absoluto es:", error_abs)

if error_abs < 10**-1:
        print(" El error absoluto es menor que 10^-1")
    else:
        print(" El error absoluto NO es menor que 10^-1")</pre>
```

```
In [4]: serie_error_abs(4)
```

```
La suma de la serie hasta 1/2^ 4 es: 1.9375
El error absoluto es: 0.0625
El error absoluto es menor que 10^-1
```

Ejercicio 2 (Bubble sort)

Corrida de escritorio

```
v_1 = [3, 2, 5, 8, 4, 1]
```

i	Vector	
0	[3, 2, 5, 8, 4, 1]	
1	[3, 2, 5, 4, 1, 8]	
2	[3, 2, 4, 1, 5, 8]	
3	[3, 2, 1, 4, 5, 8]	

•••

Resultado final:

$$v_{sorted} = [1, 2, 3, 4, 5, 8]$$

Casos de prueba:

- $v_2 = [-1, 0, 4, 5, 6, 7]$
- v_3 100_000 número aleatorios entre -200 y 145.

```
In [2]: # Caso 1
        lista1=[3, 2, 5, 8, 4, 1]
        print("Caso 1:")
        print("Lista original:", lista1)
        print("Lista ordenada:", bubble_sort(lista1.copy()))
        print()
       Caso 1:
       Lista original: [3, 2, 5, 8, 4, 1]
       El contador es: 8
       Lista ordenada: [1, 2, 3, 4, 5, 8]
In [3]: # Caso A
        listaA = [-1, 0, 4, 6, 7]
        print("Caso A:")
        print("Lista original:", listaA)
        print("Lista ordenada:", bubble_sort(listaA.copy()))
        print()
       Caso A:
       Lista original: [-1, 0, 4, 6, 7]
       El contador es: 0
       Lista ordenada: [-1, 0, 4, 6, 7]
In [ ]: import random
        import time
        # Caso B
        print("Caso B: 100000 números aleatorios entre -200 y 145")
        listaB = [random.randint(-200, 145) for _ in range(100000)]
        inicio = time.time()
        bubble_sort(listaB) # no imprimimos para no saturar la salida
        fin = time.time()
        print(f"Ordenamiento completado en {fin - inicio} segundos.") #La pc se muere y
```

Ejercicio 3

n	fib(n)
0	0
1	1
2	1
3	2
4	3
5	5
6	8
6	13
n = 11	?

```
In [5]: def iterativeFibo(n):
    if n == 0:
        print(0)
        return 0
    else:
        x = 0
        y = 1
        print(y, end=" ") # imprime el primer número (1)
        for i in range(1, n):
            z = x + y
            x = y
            y = z
            print(y, end=" ") # imprime cada número nuevo
        print(y, end=" ") # imprime cada número nuevo
        print() # salto de línea al final
        return y
```

```
In [6]: # Ejemplo aplicado
  resultado = iterativeFibo(11)
  print("Resultado final (y):", resultado)
```

1 1 2 3 5 8 13 21 34 55 89 Resultado final (y): 89

```
In [7]: # Ejemplo aplicado con 84
    resultado = iterativeFibo(84)
    print("Resultado final (y):", resultado)
```

1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28 657 46368 75025 121393 196418 317811 514229 832040 1346269 2178309 3524578 570288 7 9227465 14930352 24157817 39088169 63245986 102334155 165580141 267914296 43349 4437 701408733 1134903170 1836311903 2971215073 4807526976 7778742049 12586269025 20365011074 32951280099 53316291173 86267571272 139583862445 225851433717 3654352 96162 591286729879 956722026041 1548008755920 2504730781961 4052739537881 6557470 319842 10610209857723 17167680177565 27777890035288 44945570212853 72723460248141 117669030460994 190392490709135 308061521170129 498454011879264 806515533049393 1 304969544928657 2111485077978050 3416454622906707 5527939700884757 89443943237914 64 14472334024676221 23416728348467685 37889062373143906 61305790721611591 991948 53094755497 160500643816367088

Resultado final (y): 160500643816367088

```
In [8]: # Ejemplo aplicado
  resultado = iterativeFibo(1531)
  print("Resultado final (y):", resultado)
```

Resultado final (y): 407936176052377669101778911015323059541693566794692519680122 463207854422013990100626081201338987968421592147014912276452966402513511180974144 525129433779239442408519013425119983218373176872312001814049893514987716130911286 090664428422730299315959724514396175573827117595933842787346948580100247676460231 57013418593547269

Graficar!

• El valor de la serie fib(n)

• El valor del cociente

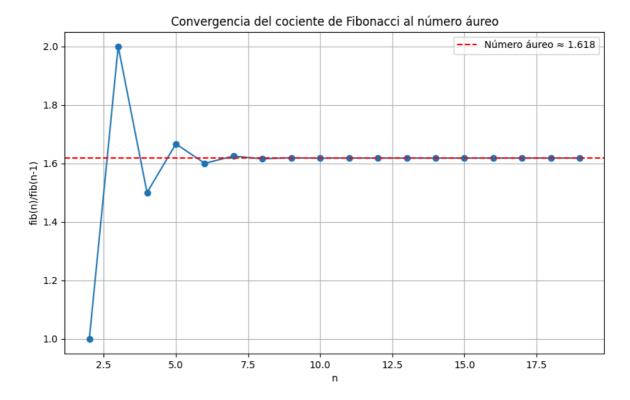
$$\phi
ightarrow rac{fib(n)}{fib(n-1)} pprox 1.618$$
 número áureo.

n	$fib(n)/fib(n \ -1)$
2	1/1 = 1
3	2/1=2
4	3/2=1.5
5	5/3 = 1.66667
6	8/5=1.6
7	13/8 = 1.625
8	21/13 = 1.615
∞	$rac{1+\sqrt{5}}{2}pprox 1.618$ (número áureo)

```
In [11]: import matplotlib.pyplot as plt
         # Función para generar los n primeros números de Fibonacci
         def fibonacci(n):
             fib\_seq = [0, 1]
             for i in range(2, n):
                 fib_seq.append(fib_seq[-1] + fib_seq[-2])
             return fib_seq
         # Función para graficar la convergencia al número áureo
         def plot_fibonacci_ratios(n_terms):
             fib_seq = fibonacci(n_terms)
             # Calcular los cocientes fib(n)/fib(n-1)
             ratios = [fib_seq[i] / fib_seq[i-1] for i in range(2, n_terms)]
             # Graficar
             plt.figure(figsize=(10,6))
             plt.plot(range(2, n_terms), ratios, marker='o', linestyle='-')
             plt.axhline(y=1.618, color='r', linestyle='--', label='Número áureo ≈ 1.618'
             plt.title('Convergencia del cociente de Fibonacci al número áureo')
             plt.xlabel('n')
             plt.ylabel('fib(n)/fib(n-1)')
             plt.legend()
             plt.grid(True)
             plt.show()
```

plot_fibonacci_ratios(n_terms)

In [12]: $n_{terms} = 20$



EJERCICIO 4

```
In [13]: def graficar_serie_armonica(N):
             Calcula y grafica la suma parcial de la serie armónica hasta N términos.
             Parámetros:
             N (int): Número de términos de la serie armónica.
             # Calcular la serie armónica
             harmonic_sum = [0] # empezamos en 0 para que la primera suma parcial sea 0
             current sum = 0
             for n in range(1, N+1):
                 current_sum += 1/n
                  harmonic_sum.append(current_sum)
             # Graficar
             plt.figure(figsize=(10,6))
             plt.plot(range(0, N+1), harmonic_sum, linestyle='-', color='b')
             plt.title(f'Suma parcial de la serie armónica \Sigma(1/n) hasta N=\{N\}')
             plt.xlabel('Número de términos n')
             plt.ylabel('Suma parcial')
             plt.grid(True)
             plt.show()
In [14]: # Ejemplo de uso
         graficar serie armonica(1000)
```

