LA ASCENSIÓN

Juan Ricardo Escobar Corredor

Primera pregunta

A continuación, la función debe retornar una lista con todos los tipos de datos de 1997/09/2023 convertido a Sting, numero, booleano si la fecha esta correcta o no.

In [1]:
```python
from datetime import datetime

def tipos_de_datos(fecha):
    tipos = []

    # Convertir la fecha a cadena de texto
    tipos.append(str(fecha))

    # Convertir la fecha a objeto datetime para verificar su validez
    try:
        fecha_dt = datetime.strptime(fecha, '%Y/%m/%d')
        tipos.append(type(fecha_dt))
        tipos.append(True)  # Agregar True si la fecha es válida
    except ValueError:
        tipos.append(None)  # Agregar None si la fecha no es válida
        tipos.append(False)  # Agregar False si la fecha no es válida

    return tipos

# Ejemplo de uso:
fecha = "1997/09/2023"
resultados = tipos_de_datos(fecha)
print(resultados)  # Output: ['1997/09/2023', <class 'datetime.datetime'>, True]
```

```
['1997/09/2023', None, False]
```

Segunda pregunta

Use el algoritmo de burbuja para encontrar el máximo y el mínimo del siguiente diccionario: dict_1 = {a : '8' , b = '2' , c ='5 , d = '6'} Recuerde ordenar por el valor

In [2]:
```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n-1):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]

def max_min_dict(dict_1):
    # Convertir los valores del diccionario en enteros
    values = [int(value) for value in dict_1.values()]

    # Ordenar los valores usando el algoritmo de burbuja
    bubble_sort(values)

    # Encontrar el mínimo y el máximo
    min_value = values[0]
    max_value = values[-1]

    return min_value, max_value

# Diccionario de ejemplo
dict_1 = {'a': '8', 'b': '2', 'c': '5', 'd': '6'}

# Encontrar el mínimo y el máximo
minimo, maximo = max_min_dict(dict_1)

print("Mínimo:", minimo)
print("Máximo:", maximo)
```

```
Mínimo: 2
Máximo: 8
```

Tercera pregunta

Genere la secuencia de Fibonacci en donde usando bucles for, la función debe crear un Data Frame hasta n = 100 donde se identifique si la suma (n + 1) es primo o no (en booleano)

In [24]:
```python
import pandas as pd

# Función para determinar si un número es primo
def es_primo(numero):
    if numero <= 1:
        return False
    for i in range(2, int(numero**0.5) + 1):
        if numero % i == 0:
            return False
    return True

# Función para generar la secuencia de Fibonacci hasta n
def fibonacci(n):
    fib_sequence = [0, 1]
    for i in range(2, n + 1):
        fib_sequence.append(fib_sequence[i - 1] + fib_sequence[i - 2])
    return fib_sequence

# Función para generar el DataFrame con la secuencia de Fibonacci y la información sobre si la suma (n + 1) es primo
def generar_dataframe_fibonacci(n):
    secuencia_fibonacci = fibonacci(n)
    suma_es_primo = [es_primo(n + 1) for n in secuencia_fibonacci]
    numeros = list(range(100, 100 - len(secuencia_fibonacci), -1))  # Columna de números desde 100 hasta (100 - len(secuencia_fibonacci) + 1)
    df = pd.DataFrame({'Número': numeros, 'Fibonacci': secuencia_fibonacci, 'Suma (n + 1) es primo': suma_es_primo})
    return df

# Generar DataFrame hasta n = 100
dataframe_fibonacci = generar_dataframe_fibonacci(100)

# Mostrar el DataFrame
print(dataframe_fibonacci)
```

```
     Número              Fibonacci  Suma (n + 1) es primo
0       100                      0                  False
1        99                      1                   True
2        98                      1                   True
3        97                      2                   True
4        96                      3                  False
..      ...                    ...                    ...
96        4   51680708854858323072                  False
97        3   83621143489848422977                  False
98        2  135301852344706746049                  False
99        1  218922995834555169026                  False
100       0  354224848179261915075                  False

[101 rows x 3 columns]
```

Nota (Respuesta):

Función es_primo(numero): Esta función determina si un número dado es primo o no. El algoritmo utilizado es una implementación eficiente para verificar si un número es primo, iterando hasta la raíz cuadrada del número dado y comprobando si es divisible por algún número menor que su raíz cuadrada.

Función fibonacci(n): Esta función genera la secuencia de Fibonacci hasta el número n. Utiliza un bucle for para iterar desde 2 hasta n y construir la secuencia de Fibonacci basada en los dos números anteriores de la secuencia.

Función generar_dataframe_fibonacci(n): Esta función genera un DataFrame con dos columnas: 'Fibonacci', que contiene la secuencia de Fibonacci hasta n generada por la función anterior, y 'Suma (n + 1) es primo', que contiene valores booleanos indicando si la suma de cada número de Fibonacci con 1 es primo o no.

Generación del DataFrame: El DataFrame se genera correctamente utilizando la función generar_dataframe_fibonacci(100), que crea una secuencia de Fibonacci hasta el número 100 y determina si la suma de cada número de Fibonacci con 1 es primo o no.

Impresión del DataFrame: Finalmente, el DataFrame se imprime utilizando el método to_markdown() para mostrarlo en formato de tabla Markdown, lo que facilita su visualización y comprensión.

Por lo tanto:

1. El código generado determina si la suma (n + 1) de cada número de Fibonacci es primo o no utilizando la función es_primo(numero). Esta función verifica si un número dado es primo o no aplicando un algoritmo eficiente que divide el número entre todos los números desde 2 hasta su raíz cuadrada. Si el número es divisible por algún número en ese rango, entonces no es primo.

En el DataFrame resultante, la columna "Suma (n + 1) es primo" contiene valores booleanos (True o False) que indican si la suma (n + 1) correspondiente a cada número de Fibonacci en la secuencia es primo o no. Si el valor es True, significa que la suma (n + 1) es primo; si es False, significa que no es primo.

1. Por consiguiente el resultado del número 100 no es primo. Esto se puede confirmar utilizando la función es_primo(numero) que se define en el código proporcionado. Cuando se pasa el número 100 a esta función, se realiza la verificación de primalidad y se concluye que 100 no es un número primo, ya que es divisible por varios números más allá de 1 y sí mismo. Por lo tanto, la respuesta es que el número 100 no es primo.

Cuarta Pregunta

Cree una función tal que de árbol con condicionales IF donde la entrada sea un parentesco y el algoritmo de salida retorne 1 si es hijo o no es hijo del nivel superior del árbol Considere una familia de 4 abuelos,2 sobrinos, 2 padres, 1 padre desconocido, 4 hijos, 7 primos.

```
              Abuelos
             /      \
        Padres       Padres
       /     \      /     \
```

Hijos Sobrinos Padre Desconocido | Primos

In [4]:
```python
def parentesco_es_hijo(parentesco):
    # Definir lista de relaciones de padres y sus descendientes
    relaciones = {
        "Abuelos": ["Padres", "Padres"],
        "Padres": ["Hijos", "Sobrinos", "Padre Desconocido"],
        "Sobrinos": ["Primos"],
        "Padre Desconocido": [],
        "Hijos": [],
        "Primos": []
    }

    # Verificar si el parentesco es hijo o no es hijo del nivel superior
    if parentesco in relaciones["Abuelos"]:
        return 1
    elif parentesco in relaciones["Padres"]:
        return 1
    else:
        return 0

# Ejemplo de uso:
parentesco = "Hijos"
resultado = parentesco_es_hijo(parentesco)
print("Es hijo del nivel superior:", resultado)
```

```
Es hijo del nivel superior: 1
```

Quinta pregunta

Genere una variable randomica, agréguelo en una lista con los primeros 100 números randomicos con seed =

1. Obtenga la desviación estándar, promedio, mínimo y máximo de la lista- Use pandas para facilidad

In [1]:
```python
import pandas as pd
import numpy as np

# Fijar la semilla
np.random.seed(15)

# Generar 100 números aleatorios y agregarlos a una lista
random_numbers = [np.random.rand() for _ in range(100)]

# Convertir la lista a un DataFrame de Pandas
df = pd.DataFrame(random_numbers, columns=['Random'])

# Calcular la desviación estándar, el promedio, el mínimo y el máximo
std_dev = df['Random'].std()
average = df['Random'].mean()
minimum = df['Random'].min()
maximum = df['Random'].max()

print("Desviación estándar:", std_dev)
print("Promedio:", average)
print("Mínimo:", minimum)
print("Máximo:", maximum)
```

```
Desviación estándar: 0.28037832010933084
Promedio: 0.44535962125232137
Mínimo: 0.0016447577600704477
Máximo: 0.998543402727172
```

Sexta pregunta

En el siguiente documento cuente la cantidad de letras usadas, genere un diccionario de salida con las frecuencias de cada letra encontrada. 'Sobre si la inteligencia artificial acabará o no quitándonos el trabajo y a dónde llegará en los próximos 10 años hay mucho escrito. Desde el equipo de robótica de Google quieren que los robots sean aún más autónomos, y ya andan trabajando en un modelo para que puedan escribir su propio código. Esto abre las puertas a que los robots puedan ejecutar tareas más complejas con la expresión completa de lenguajes como Python. También abre la puerta a que se comporten de forma indebida, aunque hay una forma de controlarlos. Mediante modelos de lenguaje de última generación, como PaLM, Google asegura que es posible escribir no solo código genérico, sino también código capaz de controlar las acciones de los robots. Tras programar varias instrucciones y combinarlas con estos modelos, dichozs modelos pueden generar de forma autónoma nuevo código.'

In [6]:
```python
import pandas as pd
from tabulate import tabulate

documento = '''Sobre si la inteligencia artificial acabará o no quitándonos el trabajo y a dónde llegará en los próximos 10 años
hay mucho escrito. Desde el equipo de robótica de Google quieren que los robots sean aún más autónomos,
y ya andan trabajando en un modelo para que puedan escribir su propio código. Esto abre las puertas a que
los robots puedan ejecutar tareas más complejas con la expresión completa de lenguajes como Python.
También abre la puerta a que se comporten de forma indebida, aunque hay una forma de controlarlos.
Mediante modelos de lenguaje de última generación, como PaLM, Google asegura que es posible escribir no
solo código genérico, sino también código capaz de controlar las acciones de los robots. Tras programar varias
instrucciones y combinarlas con estos modelos, dichos modelos pueden generar de forma autónoma nuevo
código.'''

# Convertir el documento a minúsculas para asegurar consistencia
documento = documento.lower()

# Inicializar un diccionario para almacenar las frecuencias de cada letra
frecuencias_letras = {}

# Iterar sobre cada letra del documento y contar su frecuencia
for letra in documento:
    if letra.isalpha():  # Verificar si el carácter es una letra
        if letra in frecuencias_letras:
            frecuencias_letras[letra] += 1
        else:
            frecuencias_letras[letra] = 1

# Convertir el diccionario a un DataFrame de pandas
df = pd.DataFrame(list(frecuencias_letras.items()), columns=['Letra', 'Frecuencia'])

# Ordenar el DataFrame por frecuencia de mayor a menor
df = df.sort_values(by='Frecuencia', ascending=False)

# Imprimir el DataFrame como tabla bonita y creativa
tabla = tabulate(df, headers='keys', tablefmt='fancy_grid', showindex=False)

print(tabla)
```

```
╒═════════╤══════════════╕
│ Letra   │   Frecuencia │
╞═════════╪══════════════╡
│ e       │           77 │
├─────────┼──────────────┤
│ o       │           77 │
├─────────┼──────────────┤
│ a       │           74 │
├─────────┼──────────────┤
│ s       │           48 │
├─────────┼──────────────┤
│ n       │           46 │
├─────────┼──────────────┤
│ r       │           46 │
├─────────┼──────────────┤
│ i       │           39 │
├─────────┼──────────────┤
│ l       │           33 │
├─────────┼──────────────┤
│ d       │           32 │
├─────────┼──────────────┤
│ c       │           31 │
├─────────┼──────────────┤
│ t       │           29 │
├─────────┼──────────────┤
│ u       │           27 │
├─────────┼──────────────┤
│ m       │           25 │
├─────────┼──────────────┤
│ p       │           19 │
├─────────┼──────────────┤
│ g       │           17 │
├─────────┼──────────────┤
│ b       │           17 │
├─────────┼──────────────┤
│ ó       │           11 │
├─────────┼──────────────┤
│ q       │            9 │
├─────────┼──────────────┤
│ y       │            7 │
├─────────┼──────────────┤
│ j       │            6 │
├─────────┼──────────────┤
│ á       │            5 │
├─────────┼──────────────┤
│ h       │            5 │
├─────────┼──────────────┤
│ f       │            4 │
├─────────┼──────────────┤
│ é       │            3 │
├─────────┼──────────────┤
│ x       │            2 │
├─────────┼──────────────┤
│ ú       │            2 │
├─────────┼──────────────┤
│ v       │            2 │
├─────────┼──────────────┤
│ ñ       │            1 │
├─────────┼──────────────┤
│ z       │            1 │
╘═════════╧══════════════╛
```

Séptima Pregunta

Cree una función de clase Carro que tenga atributos de ruedas, motor, y cantidad de asientos. luego genere tres clases hijas con tal de que sea coche, autobús y camioneta.

In [7]:
```python
class Carro:
    def __init__(self, ruedas, motor, asientos):
        self.ruedas = ruedas
        self.motor = motor
        self.asientos = asientos

class Coche(Carro):
    def __init__(self, motor, asientos=5):
        super().__init__(ruedas=4, motor=motor, asientos=asientos)

class Autobus(Carro):
    def __init__(self, motor, asientos=20):
        super().__init__(ruedas=6, motor=motor, asientos=asientos)

class Camioneta(Carro):
    def __init__(self, motor, asientos=8):
        super().__init__(ruedas=4, motor=motor, asientos=asientos)

# Ejemplo de uso
coche = Coche("Gasolina")
print("Coche - Motor:", coche.motor, "- Ruedas:", coche.ruedas, "- Asientos:", coche.asientos)

autobus = Autobus("Diesel")
print("Autobus - Motor:", autobus.motor, "- Ruedas:", autobus.ruedas, "- Asientos:", autobus.asientos)

camioneta = Camioneta("Gasolina")
print("Camioneta - Motor:", camioneta.motor, "- Ruedas:", camioneta.ruedas, "- Asientos:", camioneta.asientos)
```

```
Coche - Motor: Gasolina - Ruedas: 4 - Asientos: 5
Autobus - Motor: Diesel - Ruedas: 6 - Asientos: 20
Camioneta - Motor: Gasolina - Ruedas: 4 - Asientos: 8
```

Octava pregunta

Cree una función que calcule el factorial e identifique si el resultado es primo o no es primo.

In [8]:
```python
def factorial(numero):
    # Verificar si el número es negativo
    if numero < 0:
        return "No se puede calcular el factorial de un número negativo."

    # Caso base: factorial de 0 es 1
    if numero == 0:
        return 1

    # Calcular el factorial del número
    resultado = 1
    for i in range(1, numero + 1):
        resultado *= i

    return resultado

def es_primo(numero):
    # Verificar si el número es menor o igual a 1
    if numero <= 1:
        return False
    # Verificar si el número es divisible por algún número menor que su raíz cuadrada
    for i in range(2, int(numero**0.5) + 1):
        if numero % i == 0:
            return False
    return True

# Ejemplo de uso
numero = 5
resultado_factorial = factorial(numero)
print(f"El factorial de {numero} es: {resultado_factorial}")

if es_primo(resultado_factorial):
    print(f"El factorial de {numero} es primo.")
else:
    print(f"El factorial de {numero} no es primo.")
```

```
El factorial de 5 es: 120
El factorial de 5 no es primo.
```

Octava pregunta

De las siguientes fechas:

2020-07-08 00:00:00 2021-10-10 00:00:00 2020-03-30 00:00:00 2019-10-02 00:00:00 2020-03-12 00:00:00 2019-06-13 00:00:00 2018-12-12 00:00:00 2022-02-10 00:00:00 2019-06-06 00:00:00 2021-03-28 00:00:00 Obtenga a la fecha de realizado el trabajo la diferencia en días, entregue un diccionario donde la fecha sea la clave y la diferencia el valor

In [9]:
```python
from datetime import datetime
from tabulate import tabulate

def calcular_diferencia_en_dias(fecha):
    fecha_referencia = datetime(2024, 3, 12)  # Fecha de referencia
    diferencia = fecha_referencia - fecha
    return diferencia.days

fechas = [
    datetime(2020, 7, 8),
    datetime(2021, 10, 10),
    datetime(2020, 3, 30),
    datetime(2019, 10, 2),
    datetime(2020, 3, 12),
    datetime(2019, 6, 13),
    datetime(2018, 12, 12),
    datetime(2022, 2, 10),
    datetime(2019, 6, 6),
    datetime(2021, 3, 28)
]

diferencias_en_dias = {}

for fecha in fechas:
    diferencia = calcular_diferencia_en_dias(fecha)
    diferencias_en_dias[fecha.strftime('%Y-%m-%d')] = diferencia

# Convertir el diccionario a una lista de tuplas para ordenar por fecha
diferencias_en_dias_ordenadas = sorted(diferencias_en_dias.items(), key=lambda x: x[0])

# Imprimir la tabla de resultados
print("Diferencia en días para cada fecha:")
print(tabulate(diferencias_en_dias_ordenadas, headers=['Fecha', 'Diferencia en días'], tablefmt='fancy_grid'))
```

```
Diferencia en días para cada fecha:
╒════════════╤══════════════════════╕
│ Fecha      │   Diferencia en días │
╞════════════╪══════════════════════╡
│ 2018-12-12 │                 1917 │
├────────────┼──────────────────────┤
│ 2019-06-06 │                 1741 │
├────────────┼──────────────────────┤
│ 2019-06-13 │                 1734 │
├────────────┼──────────────────────┤
│ 2019-10-02 │                 1623 │
├────────────┼──────────────────────┤
│ 2020-03-12 │                 1461 │
├────────────┼──────────────────────┤
│ 2020-03-30 │                 1443 │
├────────────┼──────────────────────┤
│ 2020-07-08 │                 1343 │
├────────────┼──────────────────────┤
│ 2021-03-28 │                 1080 │
├────────────┼──────────────────────┤
│ 2021-10-10 │                  884 │
├────────────┼──────────────────────┤
│ 2022-02-10 │                  761 │
╘════════════╧══════════════════════╛
```

Decima Pregunta

Cree una calculadora básica a partir de funciones donde contenga: Sumar Restar Dividir Multiplicar Obtener la Media Obtener la desviación estándar Calcular la banda de Bollinger Calcular el mínimo Calcular el máximo Note que la calculadora no debe permitir operaciones ilógicas y debe tener una entrada para una lista de 10 dígitos en caso de calcular medidas estadísticas. Solo use funciones no use ninguna interfaz gráfica.

In [1]:
```python
import tkinter as tk
import math

def on_click(num):
    current = entry.get()
    entry.delete(0, tk.END)
    entry.insert(tk.END, current + num)

def clear():
    entry.delete(0, tk.END)

def erase():
    entry.delete(len(entry.get()) - 1)

def calculate():
    try:
        expression = entry.get()
        result = eval(expression)
        entry.delete(0, tk.END)
        entry.insert(tk.END, str(result))
    except Exception as e:
        entry.delete(0, tk.END)
        entry.insert(tk.END, "Error")

def square_root():
    try:
        num = float(entry.get())
        result = math.sqrt(num)
        entry.delete(0, tk.END)
        entry.insert(tk.END, str(result))
    except Exception as e:
        entry.delete(0, tk.END)
        entry.insert(tk.END, "Error")

def sine():
    try:
        num = float(entry.get())
        result = math.sin(math.radians(num))
        entry.delete(0, tk.END)
        entry.insert(tk.END, str(result))
    except Exception as e:
        entry.delete(0, tk.END)
        entry.insert(tk.END, "Error")

def cosine():
    try:
        num = float(entry.get())
        result = math.cos(math.radians(num))
        entry.delete(0, tk.END)
        entry.insert(tk.END, str(result))
    except Exception as e:
        entry.delete(0, tk.END)
        entry.insert(tk.END, "Error")

def tangent():
    try:
        num = float(entry.get())
        result = math.tan(math.radians(num))
        entry.delete(0, tk.END)
        entry.insert(tk.END, str(result))
    except Exception as e:
        entry.delete(0, tk.END)
        entry.insert(tk.END, "Error")

root = tk.Tk()
root.title("Advanced Calculator")

entry = tk.Entry(root, width=35, borderwidth=5)
entry.grid(row=0, column=0, columnspan=5, padx=10, pady=10)

# Define botones de los números
buttons = [
    ('1', 1, 0), ('2', 1, 1), ('3', 1, 2),
    ('4', 2, 0), ('5', 2, 1), ('6', 2, 2),
    ('7', 3, 0), ('8', 3, 1), ('9', 3, 2),
    ('0', 4, 1),
]

for (text, row, col) in buttons:
    btn = tk.Button(root, text=text, padx=20, pady=10, command=lambda t=text: on_click(t))
    btn.grid(row=row, column=col)

# Botones para operaciones
add_btn = tk.Button(root, text='+', padx=18, pady=10, command=lambda: on_click('+'))
add_btn.grid(row=1, column=3)

subtract_btn = tk.Button(root, text='-', padx=20, pady=10, command=lambda: on_click('-'))
subtract_btn.grid(row=2, column=3)

multiply_btn = tk.Button(root, text='*', padx=20, pady=10, command=lambda: on_click('*'))
multiply_btn.grid(row=3, column=3)

divide_btn = tk.Button(root, text='/', padx=20, pady=10, command=lambda: on_click('/'))
divide_btn.grid(row=4, column=3)

equal_btn = tk.Button(root, text='=', padx=20, pady=10, command=calculate)
equal_btn.grid(row=4, column=2)

clear_btn = tk.Button(root, text='AC', padx=10, pady=10, command=clear)
clear_btn.grid(row=4, column=0)

erase_btn = tk.Button(root, text='←', padx=10, pady=10, command=erase)
erase_btn.grid(row=4, column=4)

# Funciones trigonométricas
sin_btn = tk.Button(root, text='sin', padx=16, pady=10, command=sine)
sin_btn.grid(row=1, column=4)

cos_btn = tk.Button(root, text='cos', padx=14, pady=10, command=cosine)
cos_btn.grid(row=2, column=4)

tan_btn = tk.Button(root, text='tan', padx=14, pady=10, command=tangent)
tan_btn.grid(row=3, column=4)

root.mainloop()
```

In [ ]: