VISUALIZADOR DE ESTRUCTURAS DE DATOS

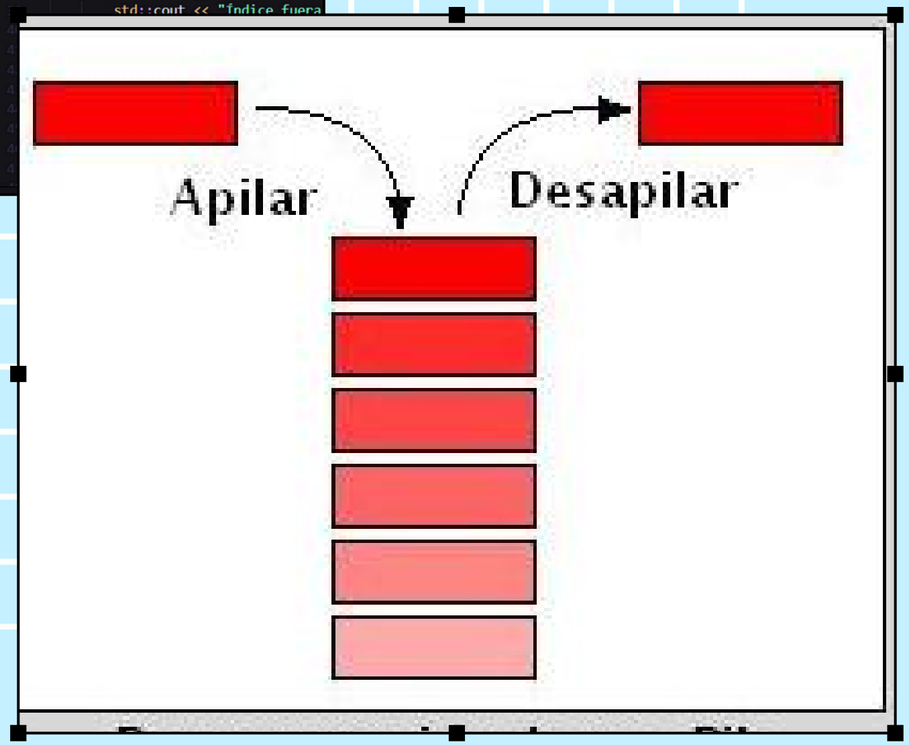
Presentado por:
- Manuel Arturo Fajardo
- Andrea Alejandra Suárez

OBJETIVO

El objetivo del proyecto es ofrecer una ayuda visual a la hora de trabajar con las diferentes estructuras de datos y las operaciones a las que se puden someter.

```
temp_usuario.cpp U   arreglo_visual.py   cola_visual.py   array.cpp
erfaz.py > VisualizadorEstructuras > ejecutar_operacion_desde_boton
VisualizadorEstructuras: # se crea la clase principal
f mostrar_feedback_operacion(self, mensaje):

  # Mensaje más atractivo
  self.canvas_visual.create_text(340, 200,
                                 text=f"✅ {mensaje}",
                                 font=("Arial", 14, "bold"),
                                 justify=tk.CENTER)

  # El feedback se quita después de 1.5 segundos y ejecut
  self.root.after(1500, self.ejecutar_codigo)

conectar los botones de la imagen con las funciones reale
f ejecutar_operacion_desde_boton(self, operacion):
  print(f"DEBUG: Botón {operacion} presionado")
  self.mostrar_feedback_operacion_btn(operacion)

  if operacion == "AGREGAR":
      self.operacion_agregar()
  elif operacion == "ELIMINAR":
      self.operacion_eliminar()
  elif operacion == "VACIAR":
      self.operacion_vaciar()

f mostrar_feedback_operacion_btn(self, operacion):
  coords_operaciones = {
      "AGREGAR": (1027, 688, 1084, 743),
      "ELIMINAR": (1116, 688, 1170, 743),
      "VACIAR": (1204, 688, 1260, 743)
  }

  if hasattr(self, 'feedback_operacion'):
```

```
src > array.cpp > ...
Array::~Array() {
}

bool Array::isEmpty() {
    return size == 0;
}

int Array::getSize() {
    return size; // nos da el tama
}

// insertar algún valor en la posi
void Array::set(int index, int val
    if (!isEmpty()) { // solo si e
        if (index >= 0 && index <
            data[index] = value; /
        };
    };
};

// extraer los valores dado un ind
int Array::get(int index) {
    if (index < 0 || index >= size
        std::cout << "Índice fuera
```

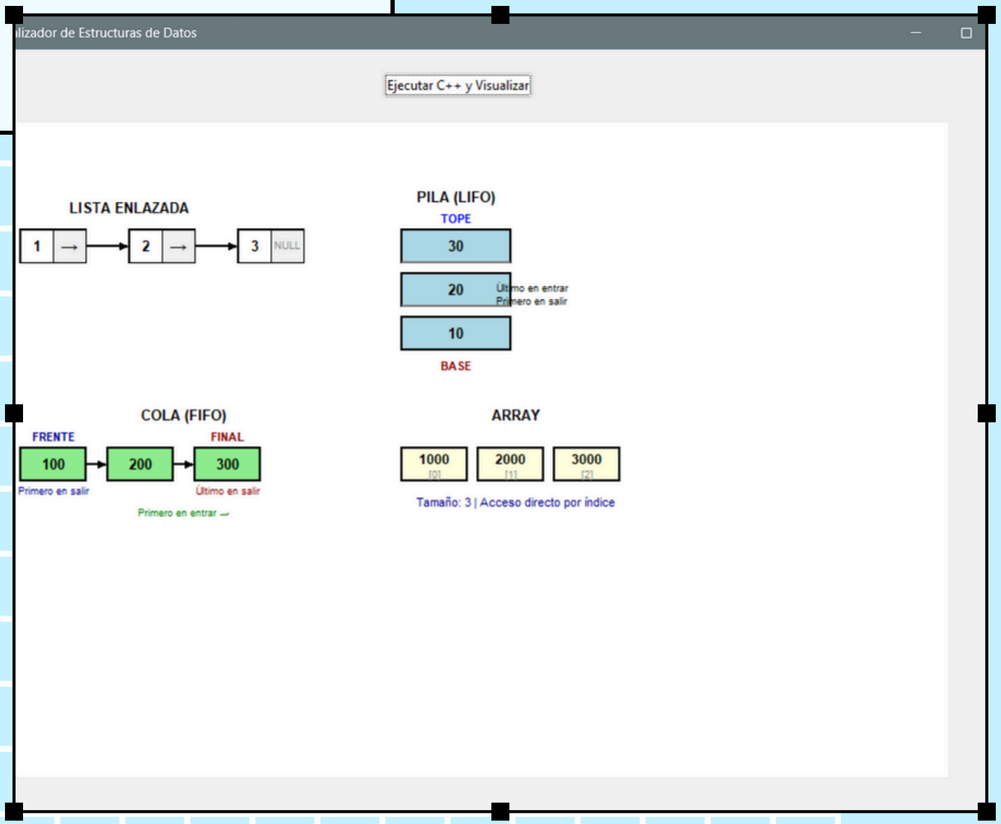


PROBLEMA

SOLUCIÓN

USUARIO

Implementación
con C++

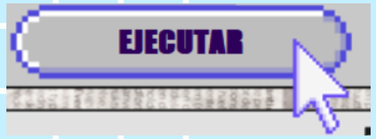


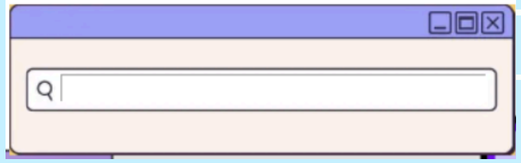
ANÁLISIS

IMPLEMENTACIÓN

INTERFAZ

Lista Pila Cola Arreglo





ESTRUCTURAS

ARREGLOS

Colección de elementos del mismo tipo almacenados en posiciones contiguas de memoria.

LISTAS

Estructura de datos formada por elementos enlazados entre sí, donde cada uno apunta al siguiente.

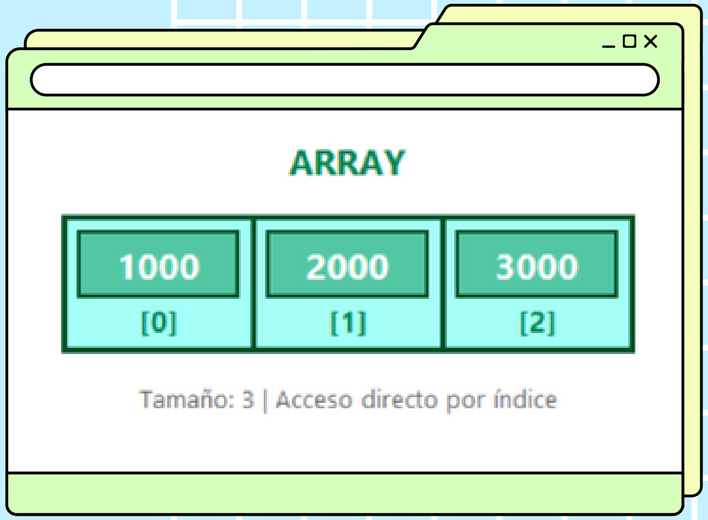
COLAS

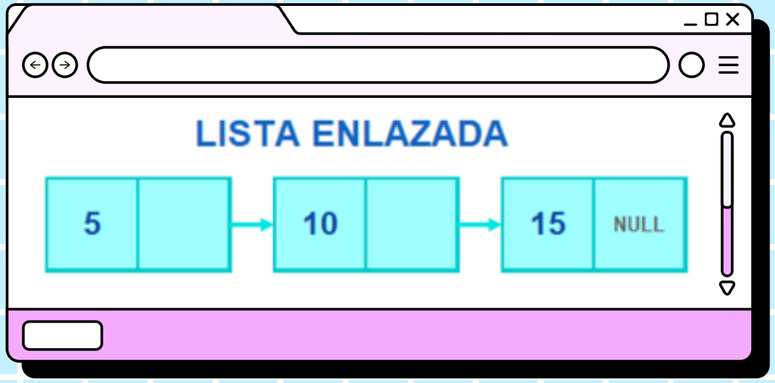
Estructura de datos donde el primer elemento en entrar es el primero en salir.

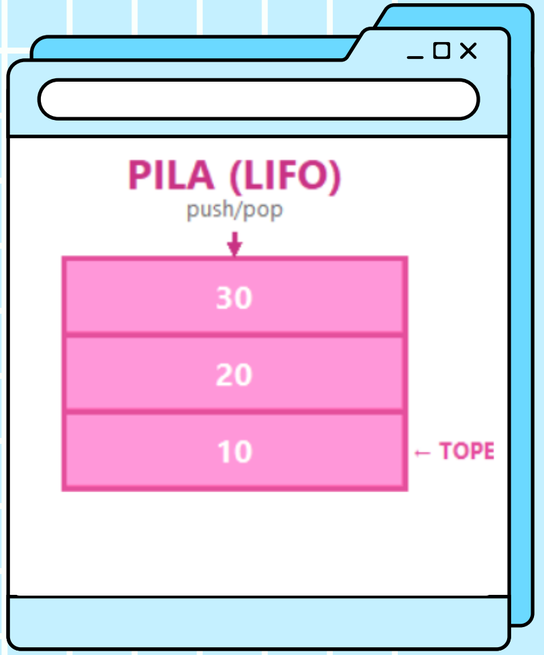
PILAS

Estructura de datos donde el último elemento en entrar es el primero en salir.

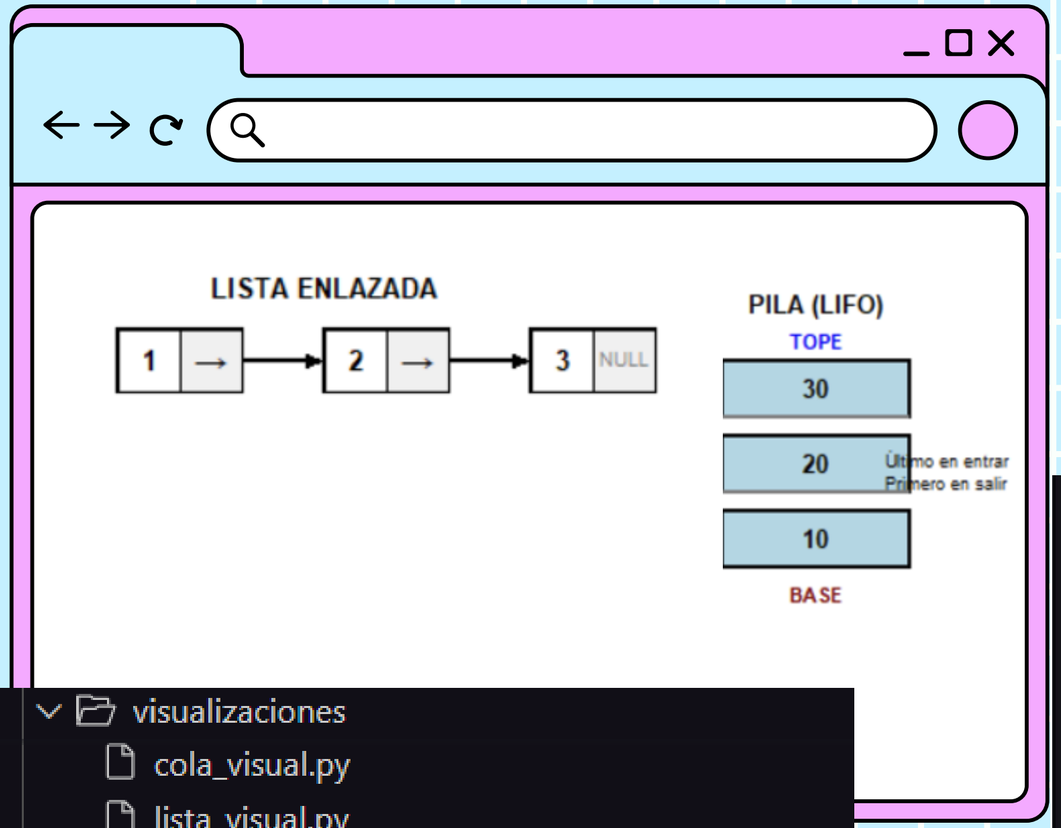
ESTRUCTURAS

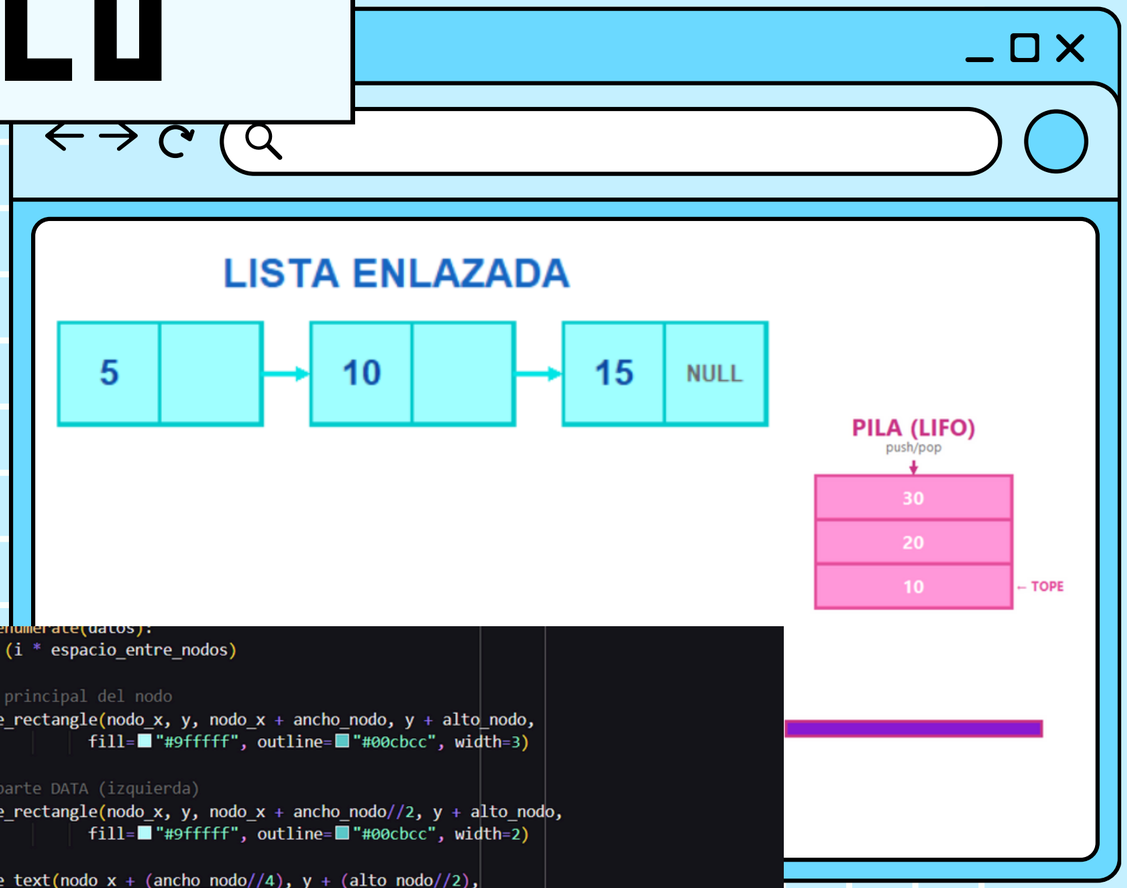




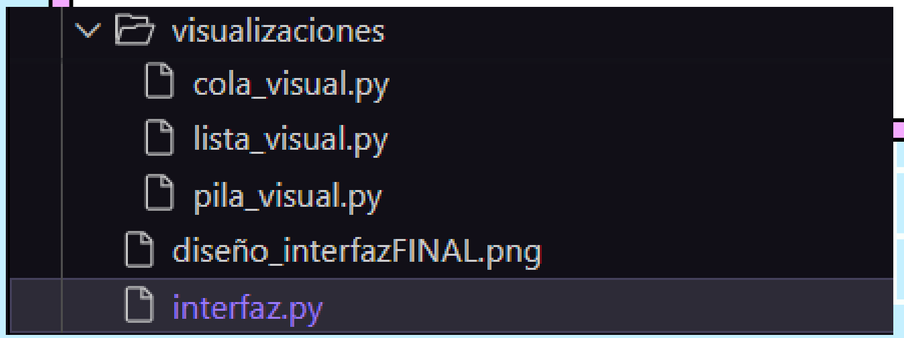


PROCESO GRÁFICO





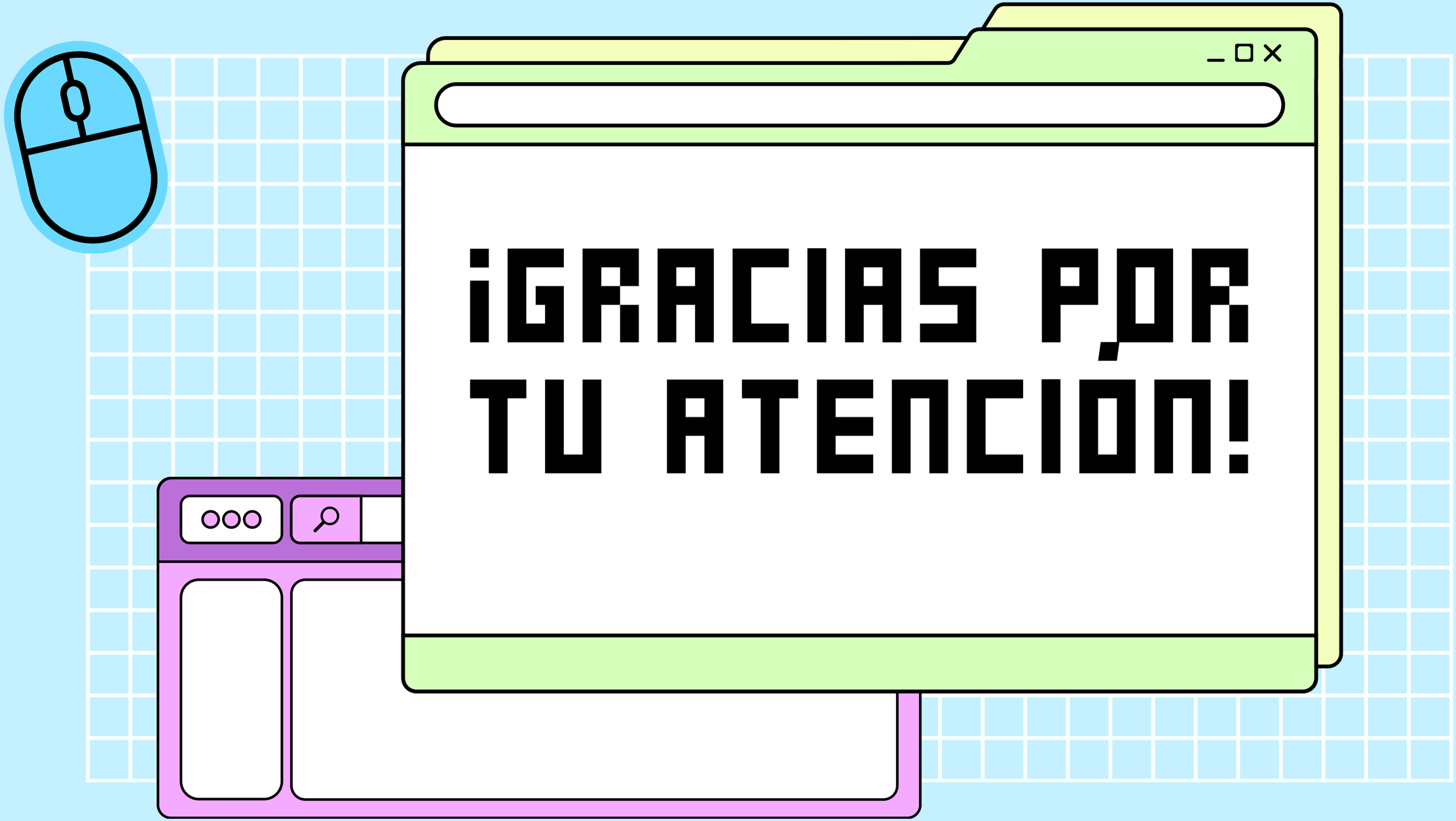

```
for i, valor in enumerate(datos):
    nodo_x = x + (i * espacio_entre_nodos)

    # rectángulo principal del nodo
    canvas.create_rectangle(nodo_x, y, nodo_x + ancho_nodo, y + alto_nodo,
                            fill="#9fffff", outline="#00cbcc", width=3)

    # división: parte DATA (izquierda)
    canvas.create_rectangle(nodo_x, y, nodo_x + ancho_nodo//2, y + alto_nodo,
                            fill="#9fffff", outline="#00cbcc", width=2)

    canvas.create_text(nodo_x + (ancho_nodo//4), y + (alto_nodo//2),
                       text=str(valor), font=("Arial", 16, "bold"), fill="#0D47A1")

    # división: parte NEXT (derecha)
    canvas.create_rectangle(nodo_x + ancho_nodo//2, y, nodo_x + ancho_nodo, y + alto_nodo,
                            fill="#9fffff", outline="#00cbcc", width=2)

    # flecha o NULL
```



VISUALIZACIONES

INTERFAZ INTUITIVA

RESULTADOS


¡GRACIAS POR
TU ATENCIÓN!