INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

Implementación de métodos computacionales (Gpo 820)

Prof. Román Martínez Martínez

José Carlos Zertuche de la Cruz

A01198177

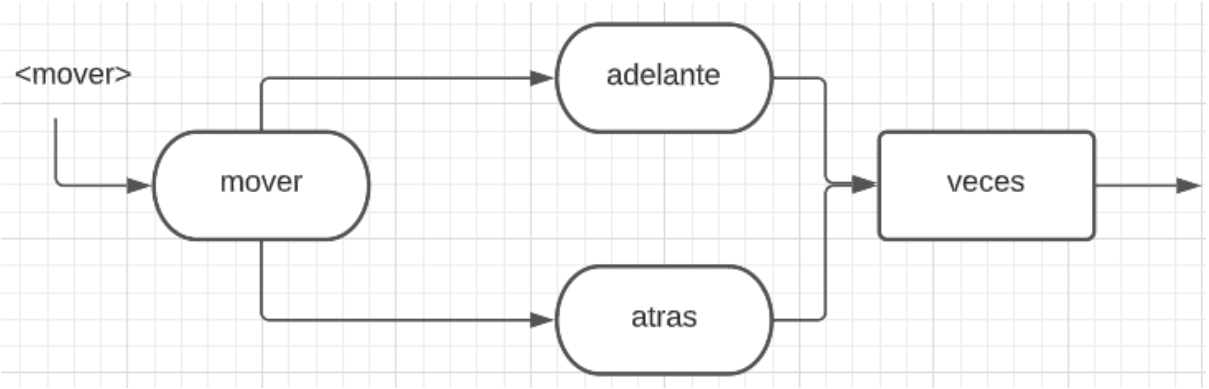
20 de marzo del 2022

Evidencia #1: Diseño e implementación básica de un DSL para enseñar a programar a niños

Elementos del léxico del lenguaje y diagramas de sintaxis

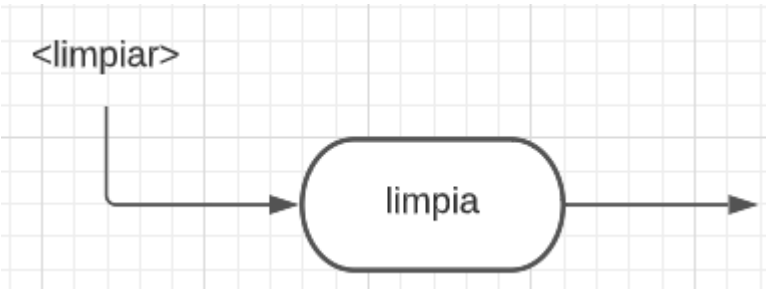
Mover

- **Expresión regular**: m_(adelante | atras)_(\[0-50]\)
- **Lexemas**: “m”, “adelante”, “atras”
- **Ejemplo**: m_atras_[20]
    - Quiere decir que se mueva 20 veces (o pasos) para atrás.



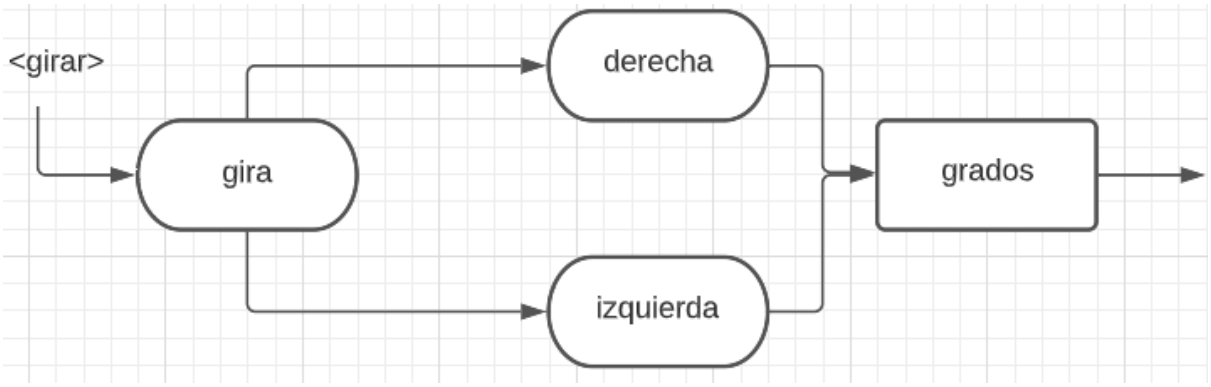
Limpiar

- **Expresión regular**: limpia
- **Lexemas**: “limpia”
- **Ejemplo**: limpia
    - Borra toda la pantalla



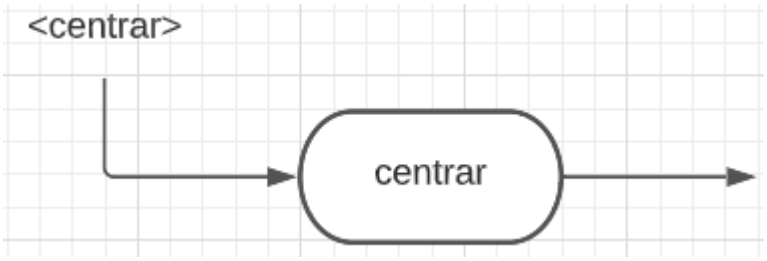
Girar

- **Lexemas**: “gira”, “der”, “izq”
- **Expresión regular**: gira_(izq | der)_(\[0-360]\)
- **Ejemplo**: gira_izq_[180]
    - Quiere decir que el objeto gire 180 grados a la izquierda, NO se mueve, solo gira su orientación.



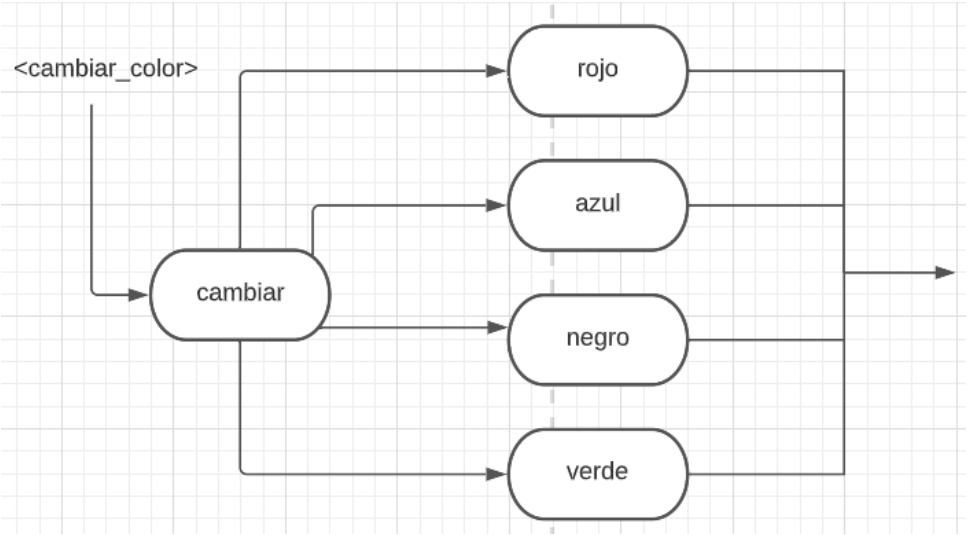
Centrar

- **Lexemas**: “centrar”
- **Expresión regular**: centrar
- **Ejemplo**: centrar
    - Pondrá al robot en medio de la pantalla



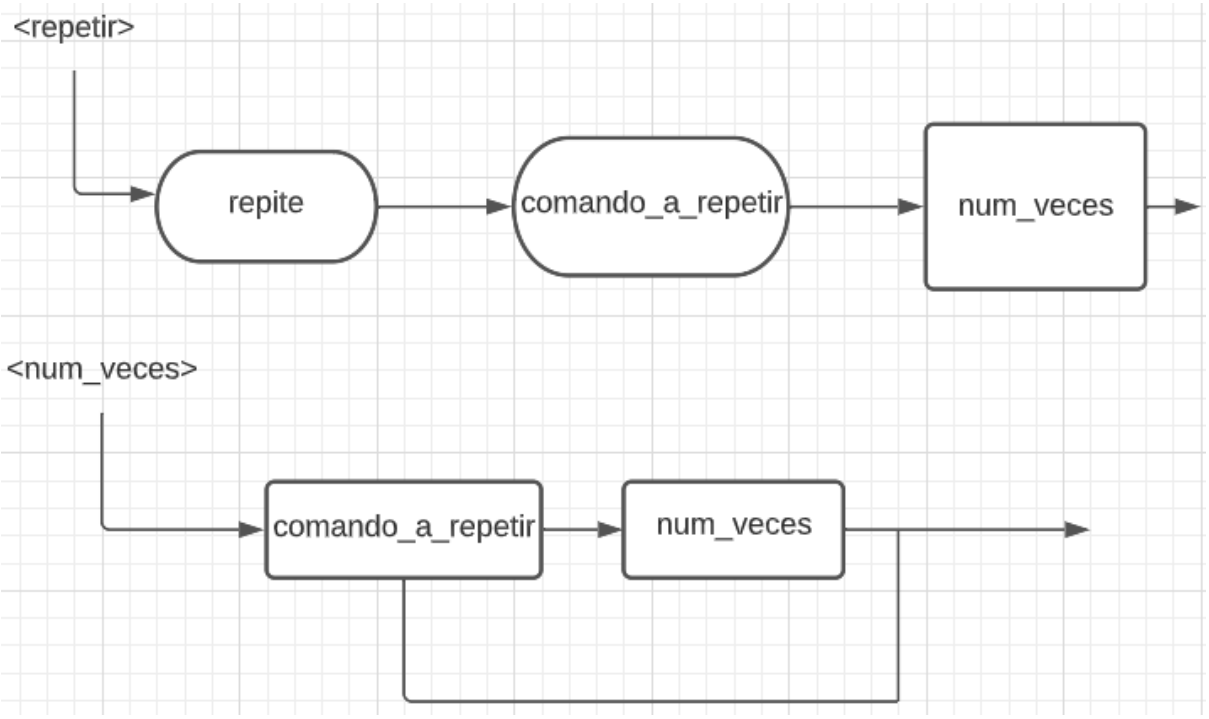
Cambiar color pluma

- **Lexemas**: “cambiar”, “rj”, “az”, “ver”, “ng”
- **Expresión regular**: cambiar_(rj | az | ver | ng)
- **Ejemplo**: cambiar_ver
    - Cambiará el color de la pluma a verde



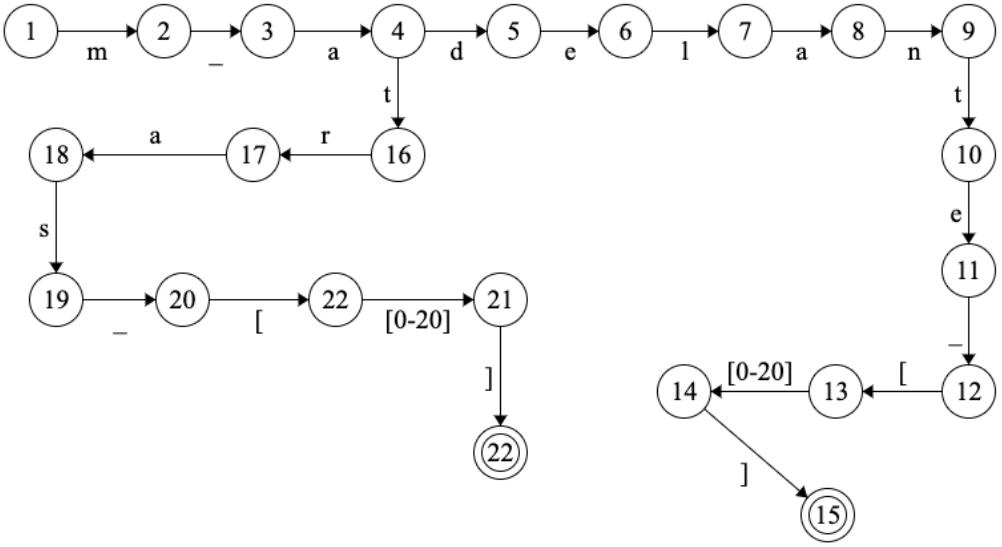
Repetir comando

- **Lexemas**: “repite”, “*comando_previamente_conocido*”
- **Expresión regular**: repite_<num_veces>_(comando_a_repetir)
- **Ejemplo**: repite_4_m_adelante_[10]
    - Repetirá 15 veces el comando mover adelante 4 unidades.

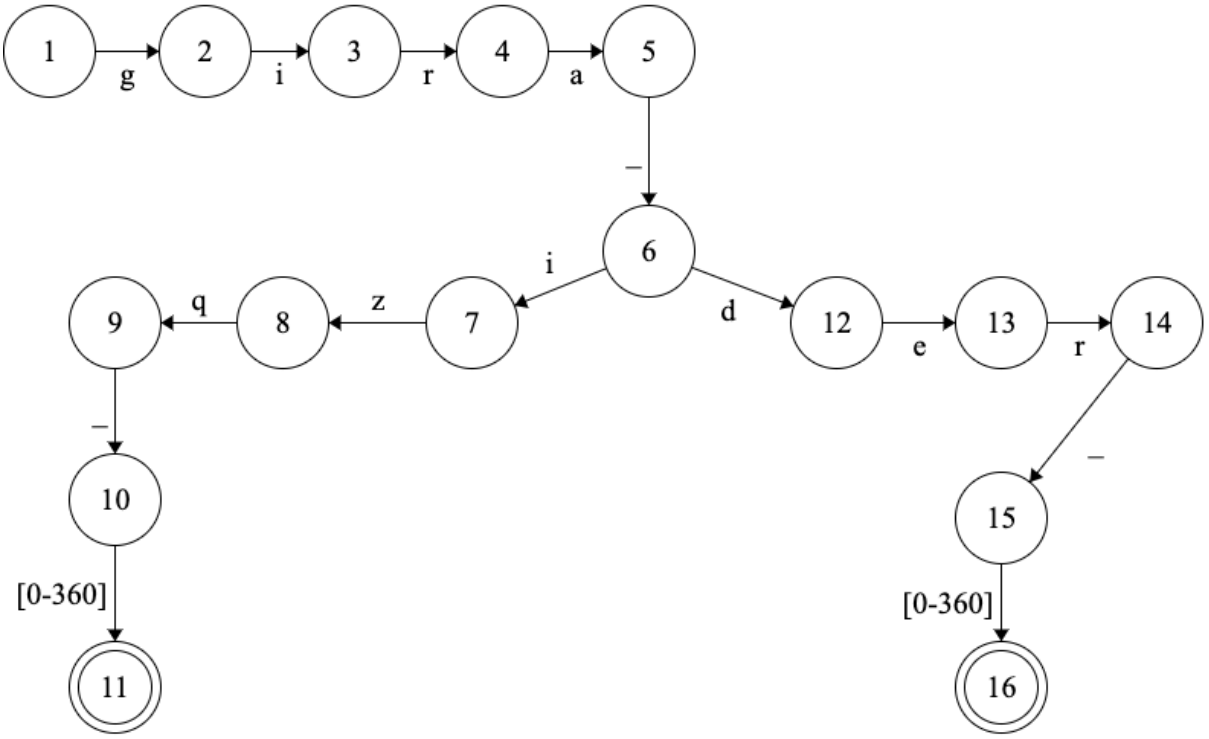


Autómata determinístico

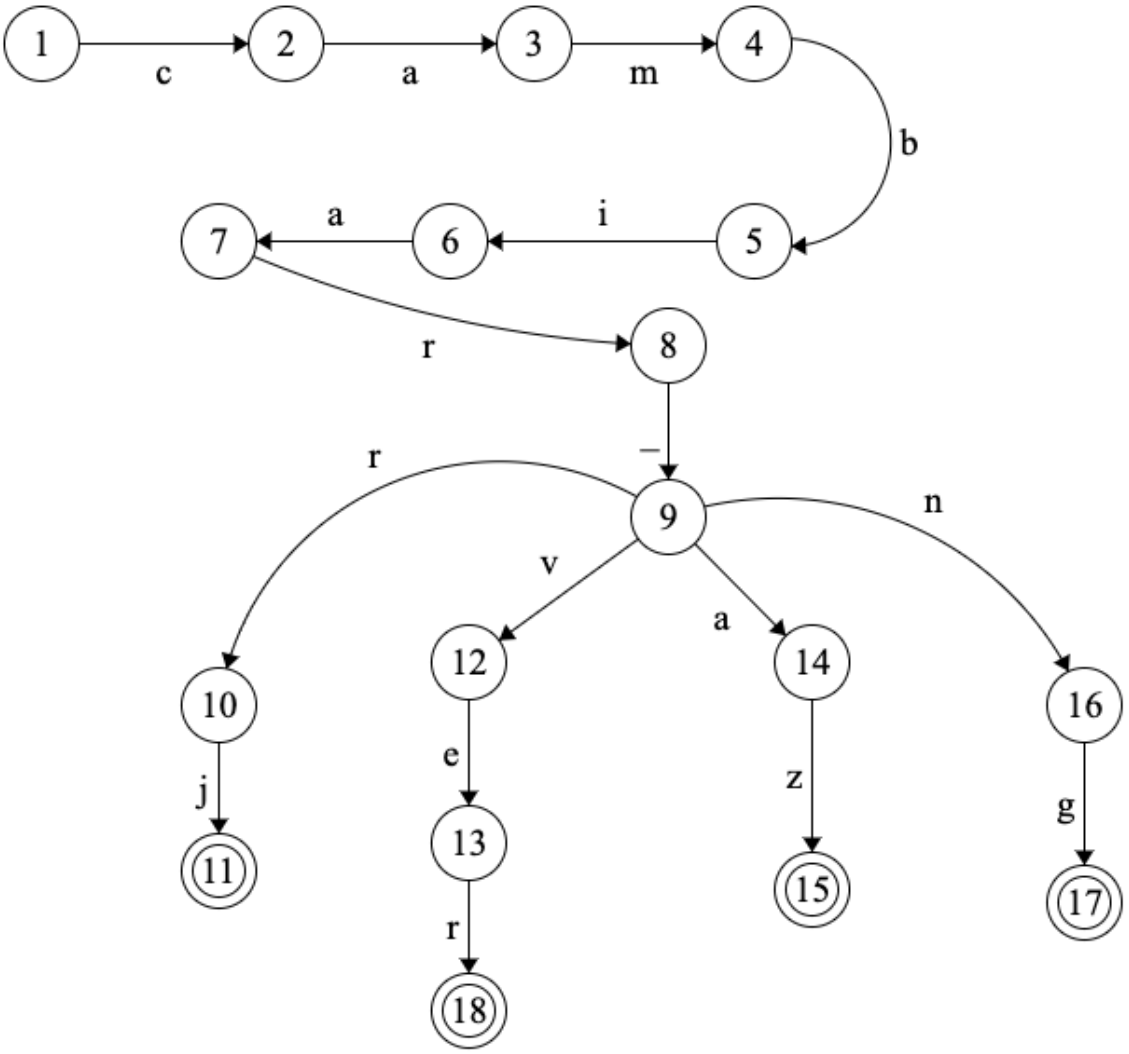
MOVER



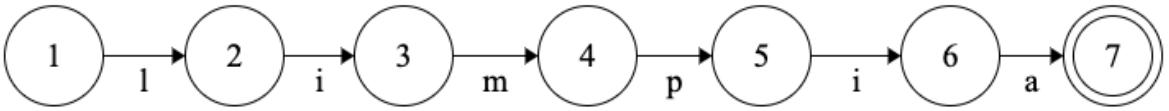
GIRAR



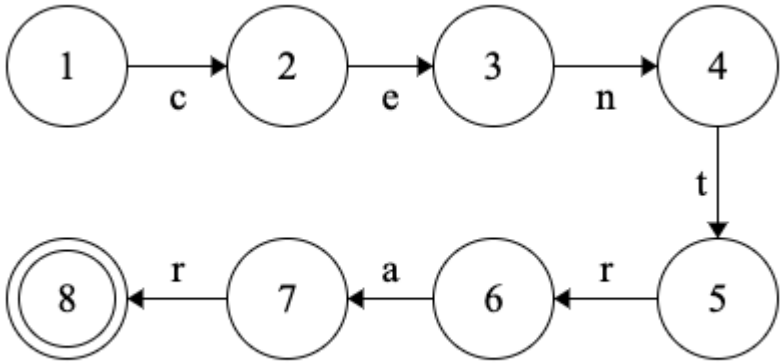
CAMBIAR COLOR DE PLUMA



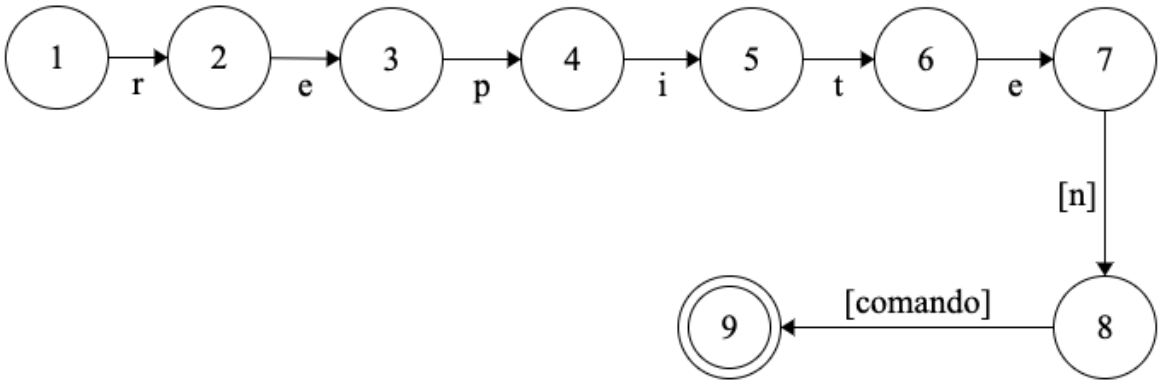
LIMPIAR LA PANTALLA



CENTRAR



REPETIR COMANDO



Gramática BNF

(Cada guión bajo, representa un espacio en blanco)

<comando> ::= <mover> | <limpiar> | <girar> | <centrar> | <cambiar_color> | <repetir>

<mover> ::= "m"_<direccion>_<distancia>

<direccion> ::= "adelante" | "atras"

<distancia> ::= "["_<numero>_"]"

<numero> ::= <digito><numero> | <digito>

<digito> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<limpiar> ::= "limpia"

<girar> ::= "gira"_<direccion_giro>_<angulo>

<direccion_giro> ::= "der" | "izq"

<angulo> ::= "["_<grados>_"]"

<grados> ::= <digito><grados> | <digito>

<centrar> ::= "centrar"

<cambiar_color> ::= "cambiar"_<color>

<color> ::= "rj" | "az" | "ver" | "ng"

<repetir> ::= "repite"_<num_veces>_<comando_a_repetir>

<num_veces> ::= <digito><num_veces> | <digito>

<comando_a_repetir> ::= <mover> | <limpiar> | <girar> | <centrar> | <cambiar_color>

Codigo implementado

```
#Autor: Jose Carlos Zertuche de la Cruz
#Evidencia 1
#Implementando un minilenguaje de programacion
#Nombre del lenguaje: Zerlang

import turtle
import re

def mover(direccion, distancia): #funcion de mover
   if direccion == "adelante":
       turtle.forward(distancia)
   elif direccion == "atras":
       turtle.backward(distancia)
```

```
def limpiar(): #funcion de limpiar la pantalla
    turtle.clear()

def girar(direccion, angulo): #funcion de girar el robot
    if direccion == "izq":
        turtle.left(angulo)
    elif direccion == "der":
        turtle.right(angulo)

def centrar(): #funcion de centrar el robot
    turtle.home()

def cambiar_color(color): #funcion de cambiar color del robot
    if color == "rj":
        turtle.pencolor("red")
    elif color == "az":
        turtle.pencolor("blue")
    elif color == "ver":
        turtle.pencolor("green")
    elif color == "ng":
        turtle.pencolor("orange")

def repetir(num_veces, comando_a_repetir): #funcion de repetir el
comando
    for i in range(num_veces):
        ejecutar_comando(comando_a_repetir)

def ejecutar_comando(token): #funcion principal utilizando el
descenso recursivo
    #Este primer if verifica si el token tiene el formato
"m_(adelante|atras)_(n)", donde n es la cantidad
    #de pasos que se movera. Si se cumple el formato, extrae la
dirección, la distancia y llama a la funcion mover
    #para mover al robot
```

```
    pasos_maximos = 50 #lo maximo que podra avanzar el robot sera
50
    if re.match(r"m_(adelante|atras)_(\[\d+\])", token):
        direccion, distancia =
re.findall(r"m_(adelante|atras)_(\[\d+\])", token)[0]
        if direccion == "adelante":
            if int(distancia[1:-1]) > pasos_maximos:
                print(f"Error: la distancia máxima para mover
hacia adelante es {pasos_maximos}")
            else:
                mover(direccion, int(distancia[1:-1]))
        elif direccion == "atras":
            if int(distancia[1:-1]) > pasos_maximos:
                print(f"Error: la distancia máxima para mover
hacia atrás es {pasos_maximos}")
            else:
                mover(direccion, int(distancia[1:-1]))
    #Este segundo if verifica si el token es igual a "limpiar"
    elif token == "limpia":
        limpiar()

    #Se verifica si tiene formato gramatical de
"gira_(izq|der)_(n)", donde n es el angulo al cual rotara el
robot.
    elif re.match(r"gira_(izq|der)_(\[\d+\])", token):
        direccion, angulo =
re.findall(r"gira_(izq|der)_(\[\d+\])", token)[0]
        girar(direccion, int(angulo[1:-1]))

    #Verifica si el token es igual a centar, y llama a la funcion
centrar
    elif token == "centrar":
        centrar()

    #Se verifica si el token es igual a "cambiar_(rj|az|ver|ng)",
si si cumple se llama a la funciond de cambiar el color.
```

```
    elif re.match(r"cambiar_(rj|az|ver|ng)", token):
        color = re.findall(r"cambiar_(rj|az|ver|ng)", token)[0]
        cambiar_color(color)

    #Se verifica si el token tiene el formato
"repite_(n)_(comando)", donde n indica la cantidad que se
repetira el comando.
    #Si se cumple el formato, guarda el número de veces, el
comando a repetir, y se llama la función repetir.
    elif re.match(r"repite_(\d+)_(.+)", token):
        num_veces, comando_a_repetir =
re.findall(r"repite_(\d+)_(.+)", token)[0]
        repetir(int(num_veces), comando_a_repetir)

#La funcion valida y ejecuta comandos dependiendo si identifica
uno de los formatos definidos en mi lenguaje,
#o caso contrario, muestra error si la sintaxis no es la correcta
o si hay algun error de por medio.
def analizar_linea(linea):
    token_valido = False
    for frase in ["m_(adelante|atras)_(\[\d+\])",
                  "limpia",
                  "gira_(izq|der)_(\[\d+\])",
                  "centrar",
                  "cambiar_(rj|az|ver|ng)",
                  "repite_(\d+)_.+"]:
        if re.match(frase, linea):
            token_valido = True
            ejecutar_comando(linea)
            break
    if not token_valido:
        print(f"Error de sintaxis: '{linea}' no es un comando
válido")

#Esto lo hice para que no se tenga que correr el programa una y
otra vez siempre.
```

```
while True:
    linea = input("Ingrese un comando: ")
    if linea == "salir":
        break
    analizar_linea(linea)

turtle.done()
```

Comentario sobre la experiencia de aprendizaje y los resultados obtenidos.

La verdad que quedé sorprendido con lo relativamente “fácil” que es hacer un lenguaje de programación ya que con las librerías de python es mucho más sencillo hacer la interfaz gráfica. Esa era mi duda. Aprendí bastante sobre el proceso detrás de un lenguaje de programación y los tipos que existen y más o menos como operan desde cero. Es un tema que me gustó mucho aunque claramente tiene su grado de complejidad, pero estoy satisfecho con la efectividad que tuve y que si me pudo funcionar correctamente. Quedo muy agradecido con los profesores que siempre nos apoyaron.

Video a youtube
https://youtu.be/fw45iAU3A4E