**THEORY OF AUTOMATA**

**Language Tokenizer**

**Project**

**Group Members**

* Muhammad Umar Farooq 22k-4218
* Timotheus Ayub 22k-4547

**Overview**

The Language Tokenizer is a Python-based tool designed to tokenize text in both English (Python) and Urdu languages. Built with simplicity and efficiency in mind, this tool separates input text into individual tokens, considering spaces as the delimiter. With a user-friendly Graphical User Interface (GUI) powered by Tkinter library, the Language Tokenizer offers ease of use for both Python developers and Urdu language enthusiasts.

**Features:**

* Dual Language Support: The tokenizer supports both Python and Urdu languages, allowing users to tokenize text in either language seamlessly.
* Tokenization: It breaks down input text into tokens, considering spaces as the primary separator. This enables efficient analysis and processing of textual data.
* Graphical User Interface (GUI): The intuitive GUI built using Tkinter library provides a user-friendly experience, allowing users to interact with the tool effortlessly.
* Python Integration: As a Python-based tool, it seamlessly integrates with Python environments, making it accessible to Python developers for text processing tasks.

**How It Works:**

* Input Text: Users input text in either Python or Urdu language into the provided text box within the GUI.
* Tokenization: Upon user input, the Language Tokenizer processes the text, separating it into individual tokens based on spaces.
* Display Tokens: The tokens are displayed either in the GUI interface or exported for further processing, depending on user preference.

Applications:

* Text Analysis: It facilitates text analysis tasks by breaking down text into smaller, analyzable units.
* Language Learning: For Urdu learners, the tokenizer can assist in understanding sentence structures by breaking down Urdu text into tokens.

**Conclusion:**

The Language Tokenizer simplifies the process of tokenizing text in both Python and Urdu languages. With its user-friendly GUI and efficient tokenization capabilities, it caters to a wide range of users, from Python developers to language enthusiasts, enabling seamless text processing and analysis tasks.

**Expected Languages**

* Python
* Urdu

**Source Code**

from tkinter import \*

from tkinter import messagebox

from tkinter import ttk

class Regex:

def \_init\_(self,pattern):

self.pattern = pattern

self.tokens = []

def match(self,text):

if self.pattern=="\w+":

i=0

k=0

print(text)

flag\_for\_single\_quotes=0

flag\_for\_double\_quotes=0

while i<len(text):

if text[i]==" " or text[i]=="." or text[i]=="(" or text[i]==")" or text[i]=="=" or text[i]=="+" or text[i]=="-" or text[i]=="\*" or text[i]=="/" or text[i]=="%" or text[i]=="&" or text[i]=="|" or text[i]=="^" or text[i]=="~" or text[i]=="!" or text[i]=="<" or text[i]==">" or text[i]=="?" or text[i]==":" or text[i]==";" or text[i]=="," or text[i]=="[" or text[i]=="]" or text[i]=="{" or text[i]=="}" or text[i]=="\\" or text[i]=="'" or text[i]=='"':

if text[i]=='"':

while text[i+1]!='"' :

i+=1

i+=1

self.tokens.append(text[k:i+1])

if flag\_for\_double\_quotes==1:

i+=1

k=i

flag\_for\_double\_quotes=0

continue

elif text[i]=="'":

while text[i+1]!="'":

i+=1

i+=1

self.tokens.append(text[k:i])

if flag\_for\_single\_quotes==1:

i+=1

k=i

flag\_for\_single\_quotes=0

continue

elif text[i]==".":

self.tokens.append(text[k:i])

self.tokens.append(".")

elif text[i]=="(":

self.tokens.append(text[k:i])

self.tokens.append("(")

elif text[i]==")":

self.tokens.append(text[k:i])

self.tokens.append(")")

elif text[i]=="=":

self.tokens.append(text[k:i])

self.tokens.append("=")

elif text[i]=="+":

self.tokens.append(text[k:i])

self.tokens.append("+")

elif text[i]=="-":

self.tokens.append(text[k:i])

self.tokens.append("-")

elif text[i]=="\*":

self.tokens.append(text[k:i])

self.tokens.append("\*")

elif text[i]=="/":

self.tokens.append(text[k:i])

self.tokens.append("/")

elif text[i]=="%":

self.tokens.append(text[k:i])

self.tokens.append("%")

elif text[i]=="&":

self.tokens.append(text[k:i])

self.tokens.append("&")

elif text[i]=="|":

self.tokens.append(text[k:i])

self.tokens.append("|")

elif text[i]=="^":

self.tokens.append(text[k:i])

self.tokens.append("^")

elif text[i]=="~":

self.tokens.append(text[k:i])

self.tokens.append("~")

elif text[i]=="!":

self.tokens.append(text[k:i])

self.tokens.append("!")

elif text[i]=="<":

self.tokens.append(text[k:i])

self.tokens.append("<")

elif text[i]==">":

self.tokens.append(text[k:i])

self.tokens.append(">")

#i+=1

elif text[i]=="?":

self.tokens.append(text[k:i])

self.tokens.append("?")

#i+=1

elif text[i]==":":

self.tokens.append(text[k:i])

self.tokens.append(":")

#i+=1

elif text[i]==";":

self.tokens.append(text[k:i])

self.tokens.append(";")

#i+=1

elif text[i]==",":

self.tokens.append(text[k:i])

self.tokens.append(",")

#i+=1

elif text[i]=="[":

self.tokens.append(text[k:i])

self.tokens.append("[")

#i+=1

elif text[i]=="]":

self.tokens.append(text[k:i])

self.tokens.append("]")

#i+=1

elif text[i]=="{":

self.tokens.append(text[k:i])

self.tokens.append("{")

#i+=1

elif text[i]=="}":

self.tokens.append(text[k:i])

self.tokens.append("}")

#i+=1

else:

self.tokens.append(text[k:i])

k=i+1

i+=1

if i==len(text):

break

if(text[i]=='"'):

flag\_for\_double\_quotes=1

continue

if(text[i]=="'"):

flag\_for\_single\_quotes=1

continue

#i+=1

self.tokens.append(text[k:])

return self.tokens

def formatting\_output(new):

new= [item.strip() for item in new if item]

new = [item for item in new if item != '\n' or new.count(item) > 1] # Remove standalone '\n'

return "\n".join(new)

def is\_empty\_source\_code(source\_code):

for char in source\_code:

if not char.isspace(): # Efficiently check for non-whitespace characters

return True

return False

def tokenize\_func():

button\_text = Tokenize\_Button.cget("text")

if button\_text=="Clear":

Source\_code\_text.delete("1.0", END) # "1.0" is the first line and "END" is the end

Source\_code\_label.configure(text="Source Code:")

Tokenize\_Button.configure(text="Tokenize")

return

if combo.get()=="":

messagebox.showerror(title="Error",message="Select Language to tokenize")

return

source\_code = Source\_code\_text.get("1.0", END) # "1.0" is the first line and "END" is the end

flag=0

for i in source\_code:

if i!="\n" or is\_empty\_source\_code(source\_code):

flag=1

break

if flag:

new=Regex("\w+").match(source\_code)

new=formatting\_output(new)

Source\_code\_text.delete("1.0", END) # "1.0" is the first line and "END" is the end

Source\_code\_text.insert(END, new)

Source\_code\_label.configure(text="Tokenized Code:")

Tokenize\_Button.configure(text="Clear")

else:

messagebox.showerror(title="Error",message="Enter some source code")

root = Tk()

root.title("Language Tokenizer")

root.geometry("500x500")

root.resizable(False, False)

Title\_Label = Label(root, text="Language Tokenizer", font=("Helvetica", 20,"bold"),fg="red").place(x=130, y=10)

SubTitle\_Label = Label(root, text="by 22K-4218 , 22K-4547", font=("Helvetica", 15,"bold")).place(x=145, y=50)

Source\_code\_label = Label(root, text="Source Code:", font=("Helvetica", 15,"bold"))

Source\_code\_label.place(x=20, y=120)

Source\_code\_text = Text(root, height=7, width=57)

Source\_code\_text.place(x=20, y=150)

Language\_input\_label = Label(root, text="Language Input: ", font=("Helvetica", 15,"bold"))

Language\_input\_label.place(x=20, y=280)

combo = ttk.Combobox(state="readonly",values=["Python","Urdu"])

combo.place(x=180, y=285)

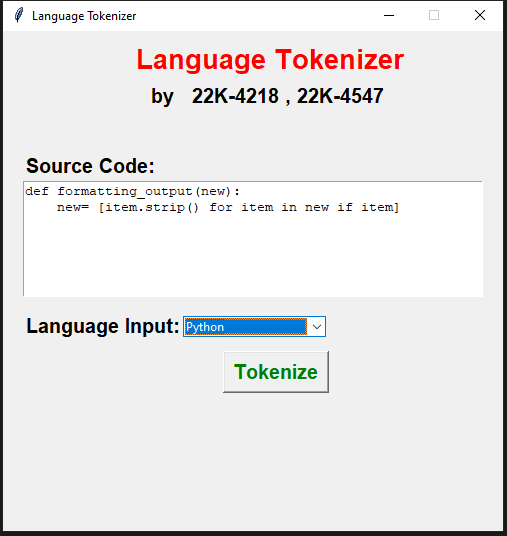
Tokenize\_Button = Button(root, text="Tokenize",command=tokenize\_func, font=("Helvetica", 15,"bold"),fg="Green",width=8,height=1)

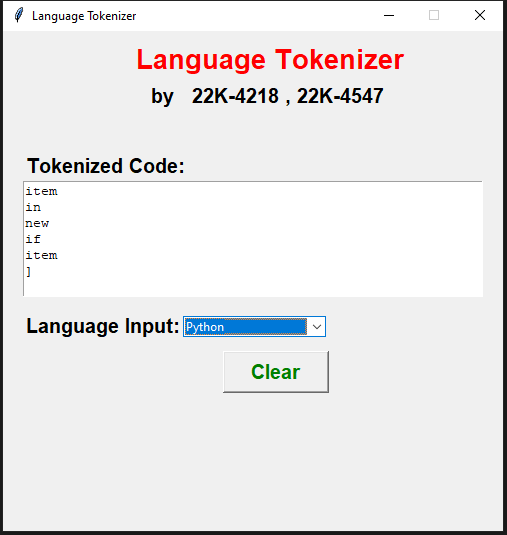
Tokenize\_Button.place(x=220, y=320)

root.mainloop()

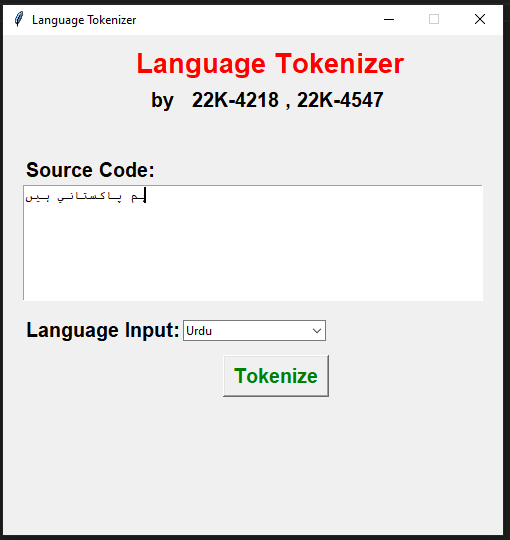
**OUTPUT**

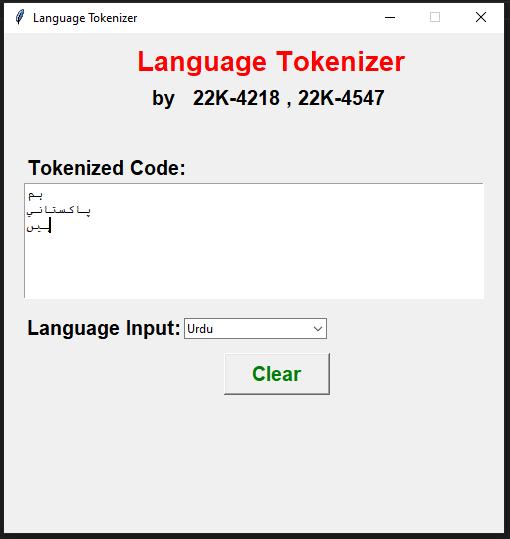
[ PYTHON]

****

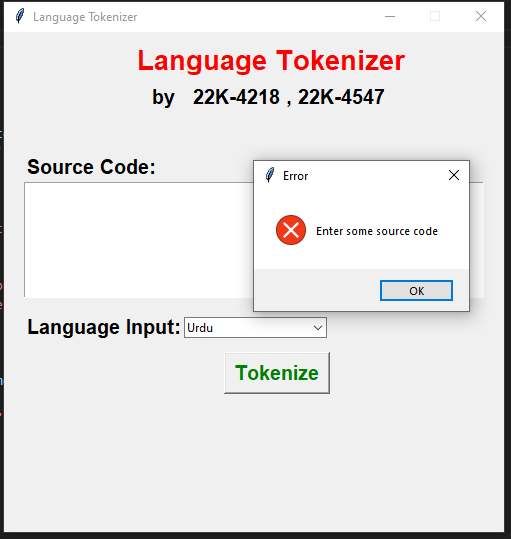
****

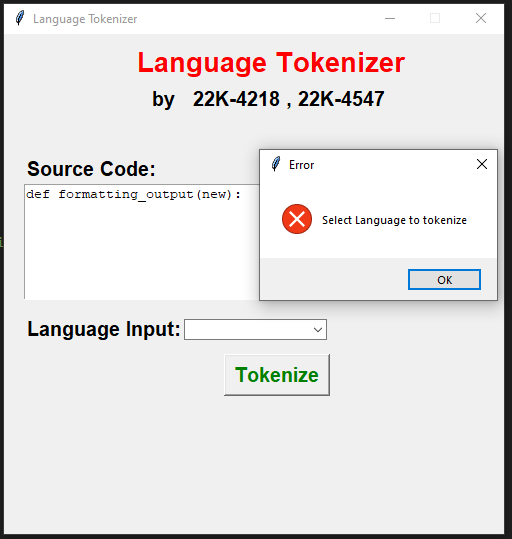
[ URDU]

****

****

[ ERROR HANDLING]

****

****