Situación Problema 1: “No sólo de los lenguajes populares vive un ITC”: aprendiendo ágilmente un pequeño lenguaje

Juan Daniel Rodríguez Oropeza A01411625

Monterrey, Nuevo León, México

Marzo 20, 2022

Ing. Román Martínez Martínez

Instituto Tecnológico y de Estudios Superiores de Monterrey

Implementación de Métodos Computacionales

El programa consta de 3 partes fundamentales:

1. **Análisis de Léxico.** Entrada: texto; salida: lexemas y tokens.
2. **Análisis de Sintaxis.** Entrada: lista de tokens; salida: nada (todo bien) error (avisa que hay error de sintaxis.
3. **Ejecutor de Música.** Entrada: lista de lexemas de notas, se desglosa y parametriza; salida: sonido.

Elementos del Léxico (Regex)

1. **Mnote =** / ((((A | B | C | D | E | F | G) (#+ | b+) *) (-2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8) | (R)) ((w | h | q | e | s | t | f) (\.+ | (t | 3 | 5 | 7 | 9)) ?) /
2. **idVozInst =** / [a-z]+ /
3. **py =** / \ | /
4. **comentarios =** / #.+ /

El resto del contenido se encuentra en las siguientes páginas.

Autómata Determinístico

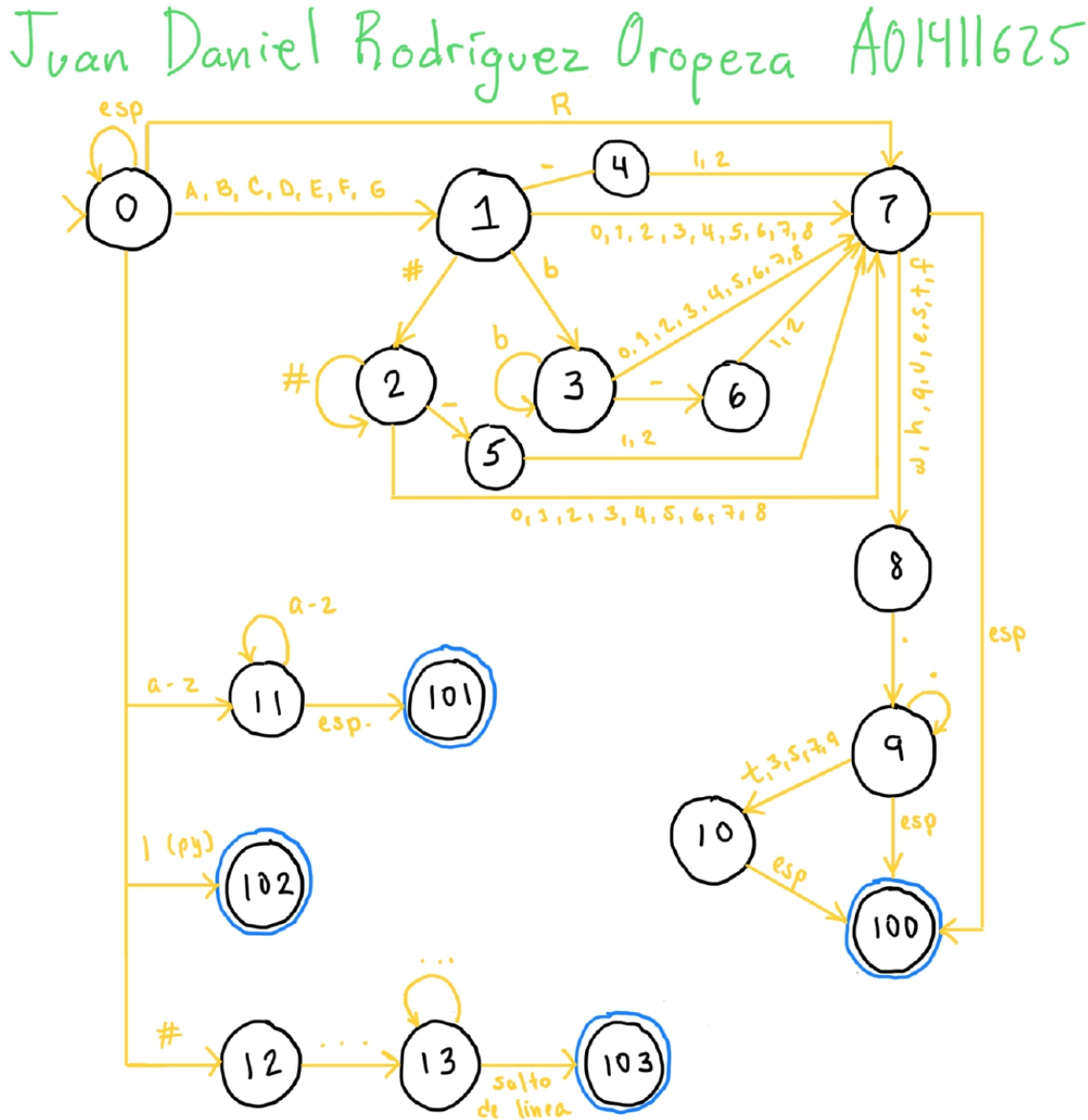
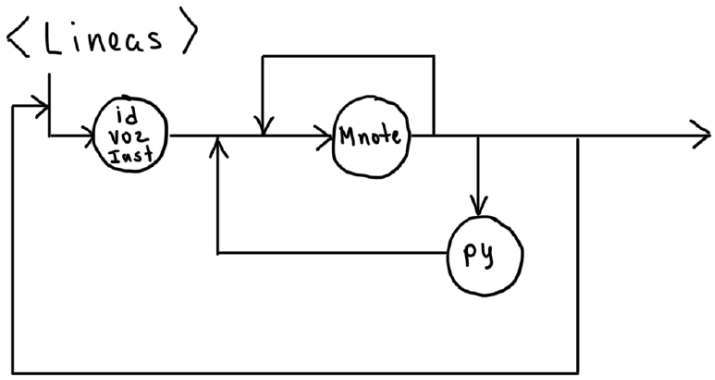


Diagrama de Sintaxis

Juan Daniel Rodríguez Oropeza A01411625



Gramática BNF

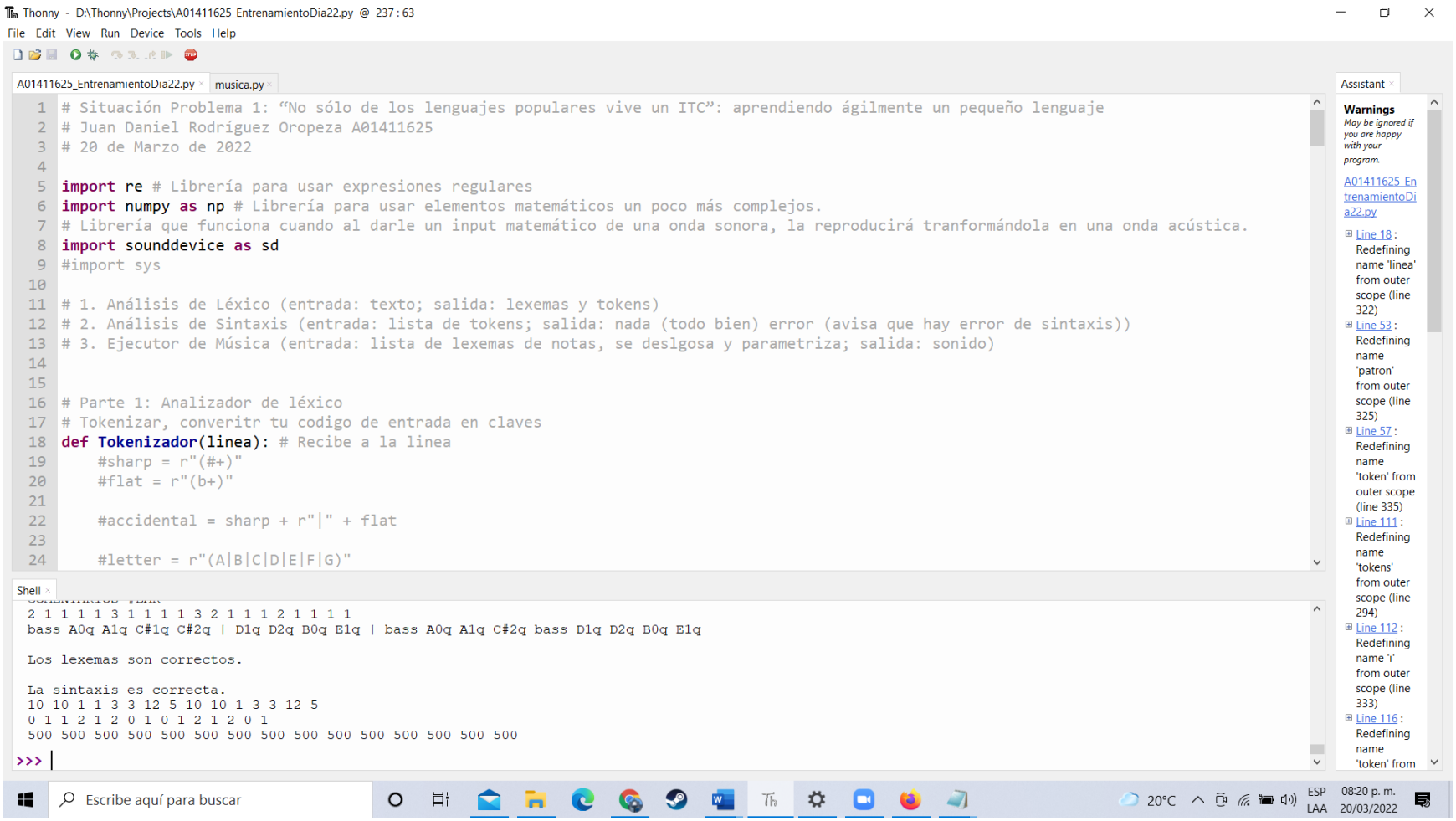
<Lineas> ::= idVozInst <Notas> | <Lineas>

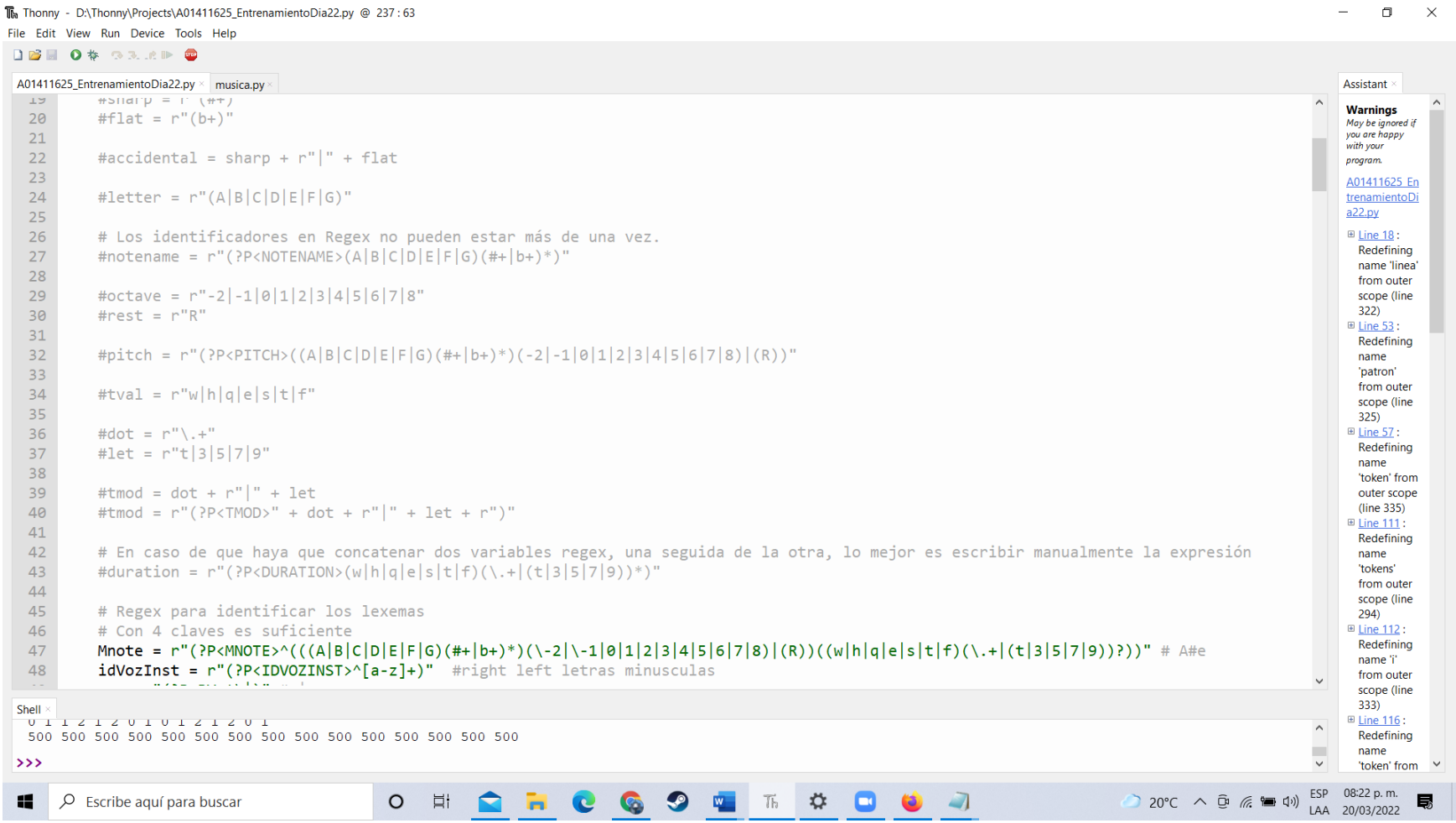
<Notas> :: = Mnote (e | py | <Notas>)

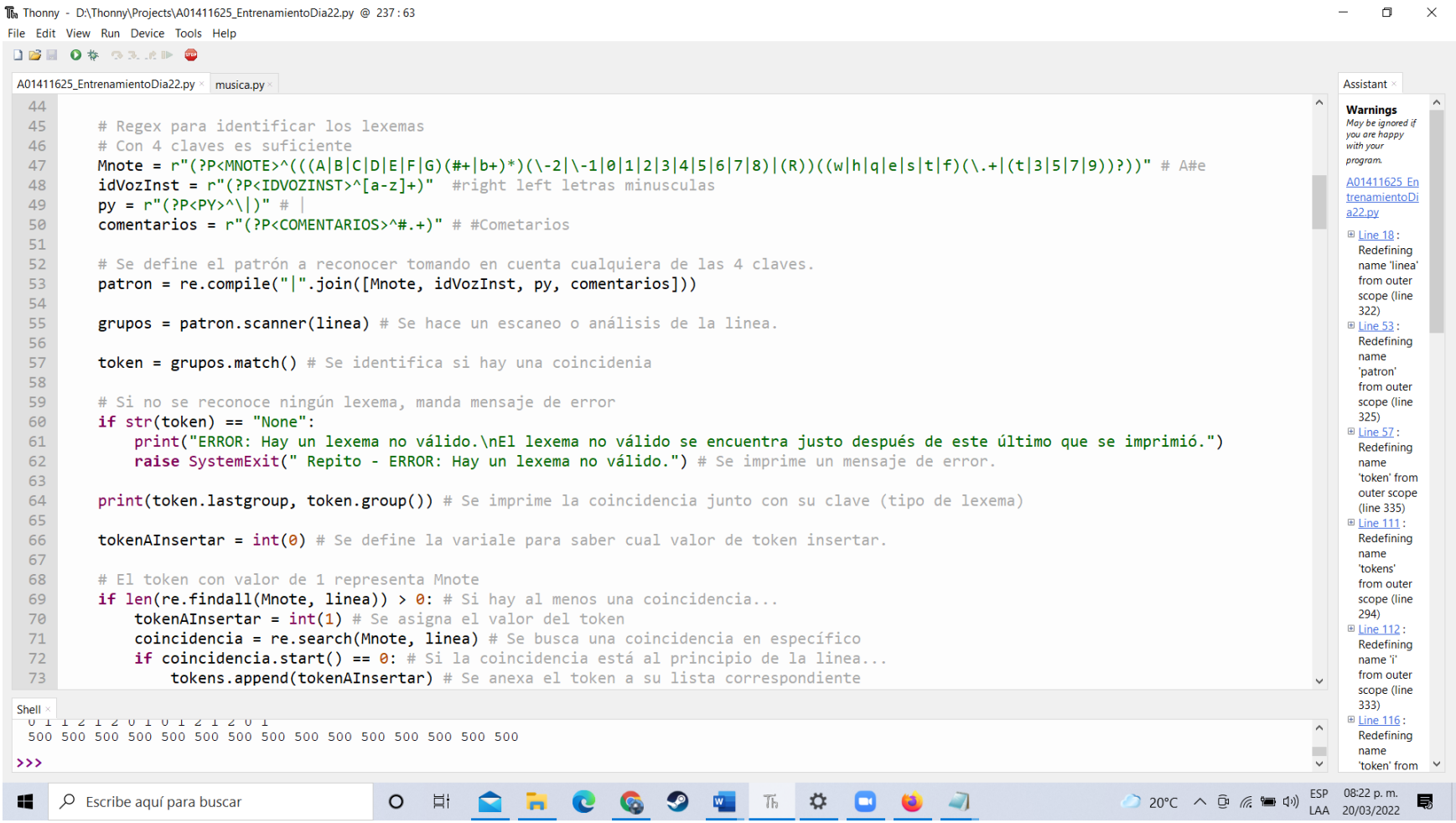
El resto del contenido se encuentra en las siguientes páginas.

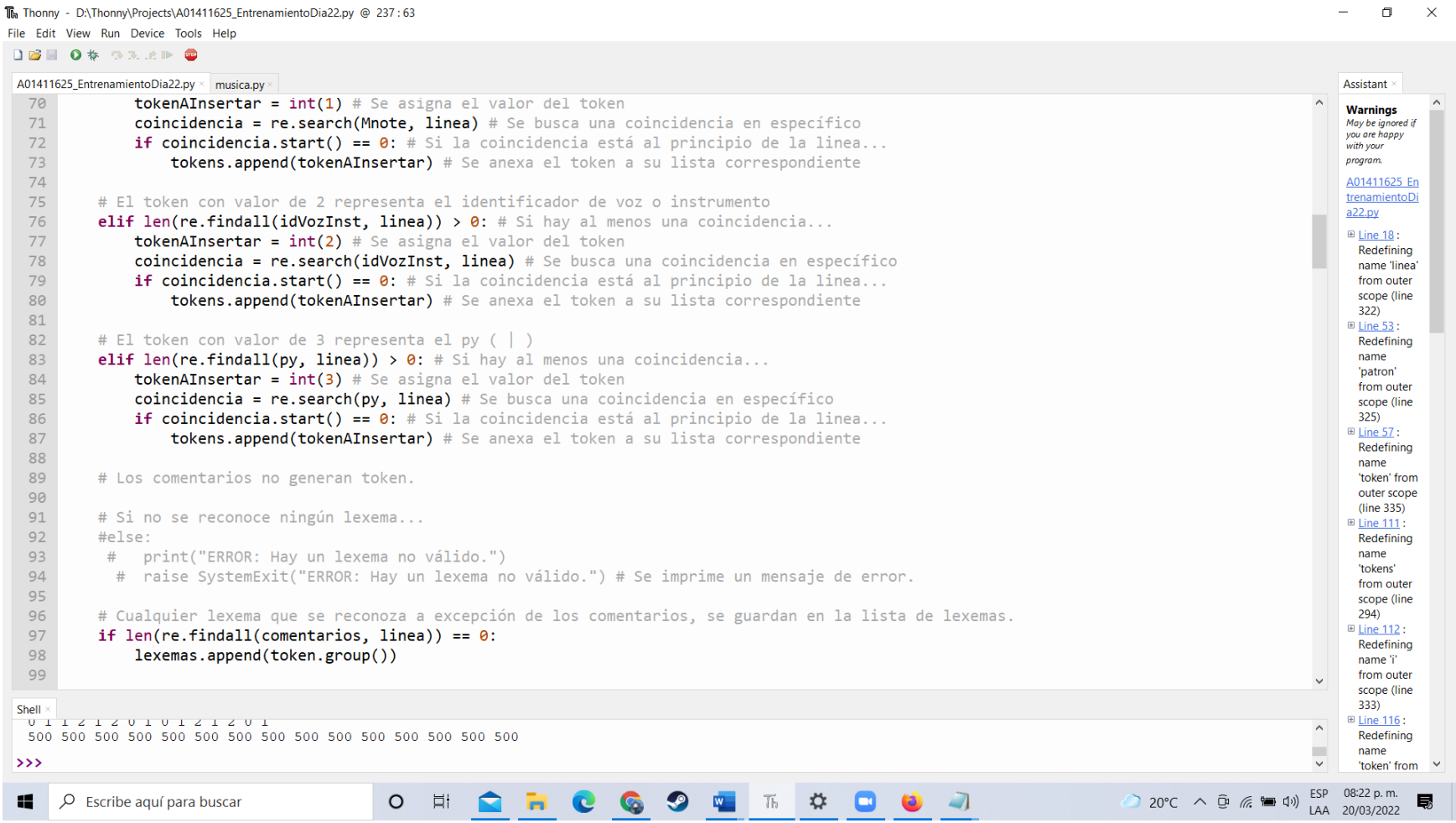
Después del código se encuentra el link del video y la reflexión.

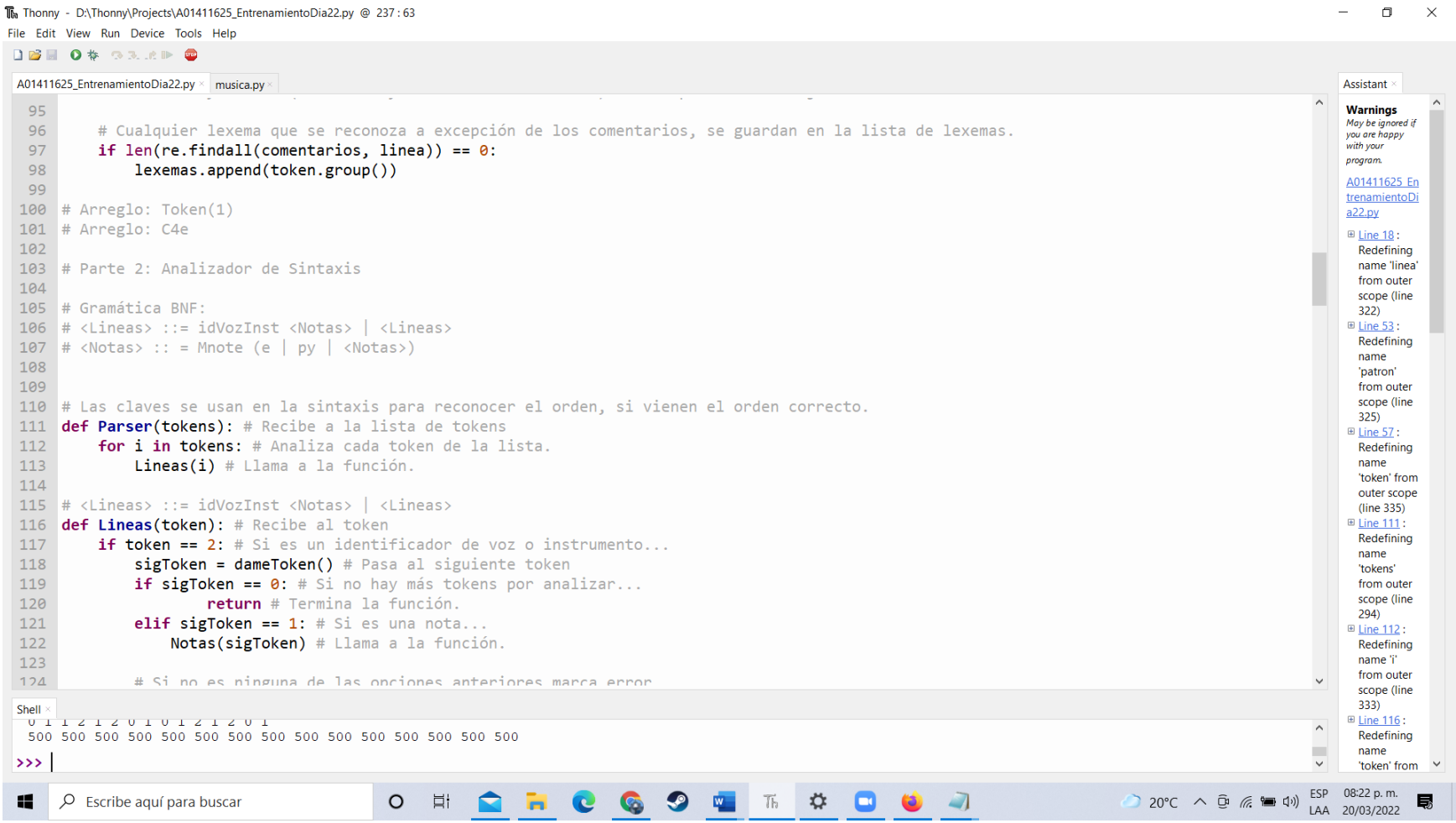
Código Implementado

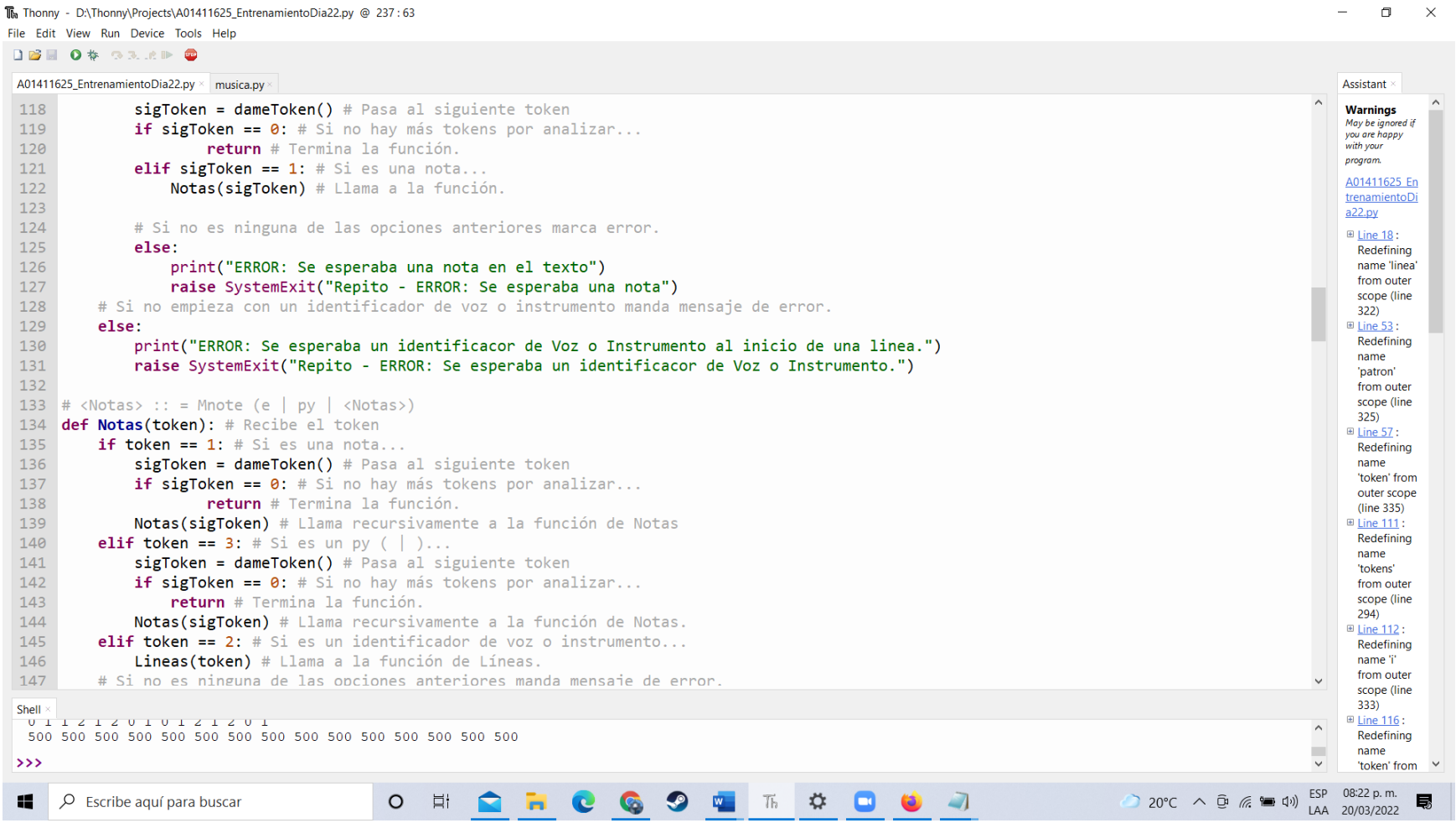


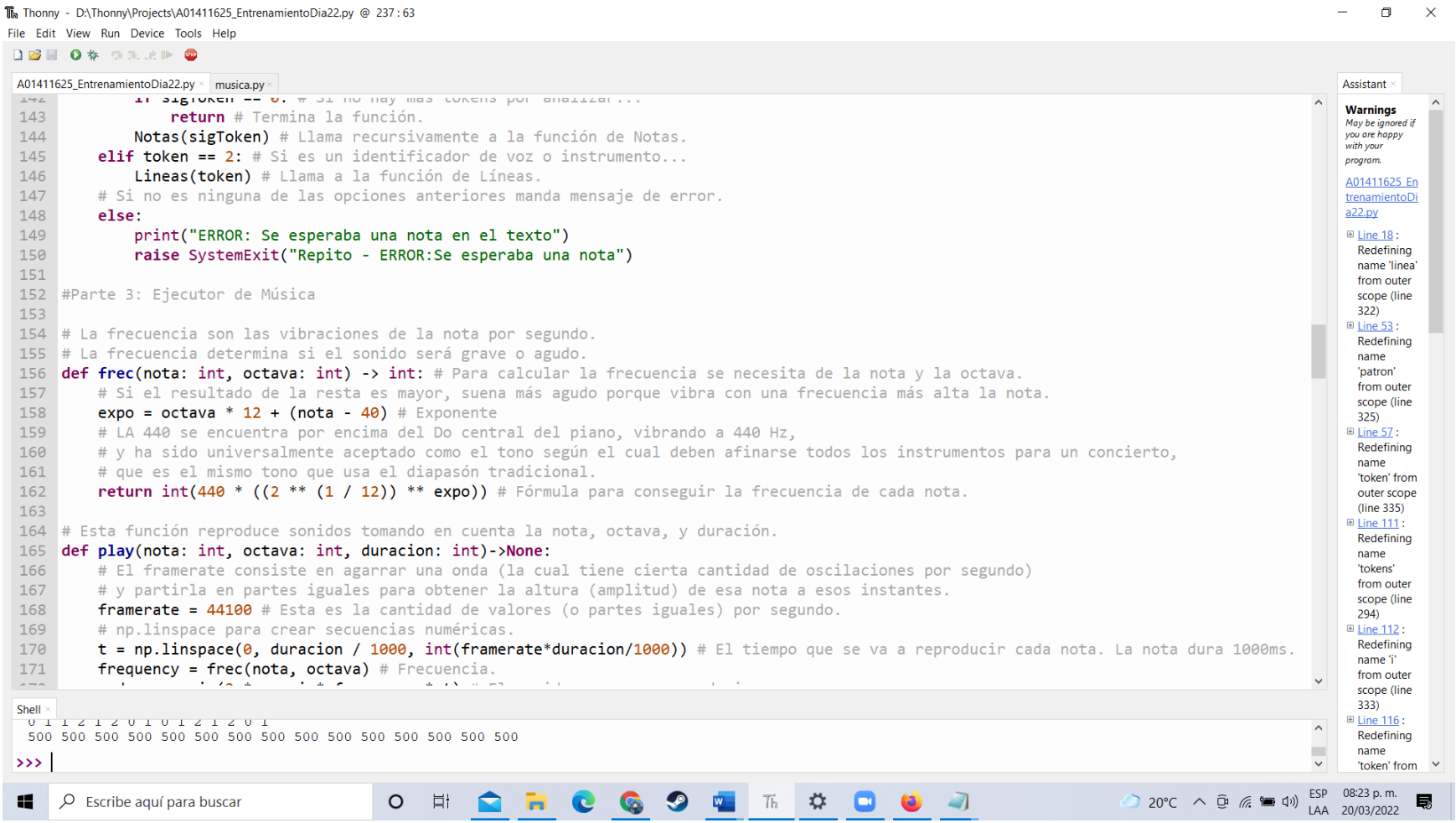

Thonny - D:\Thonny\Projects\A01411625_EntrenamientoDia22.py @ 237 : 63
File Edit View Run Device Tools Help
A01411625_EntrenamientoDia22.py
musica.py
#flat = r"(b+)"
#accidental = sharp + r"|" + flat
#letter = r"(A|B|C|D|E|F|G)"
Los identificadores en Regex no pueden estar más de una vez.
#notename = r"(?P<NOTENAME>(A|B|C|D|E|F|G)(#+|b+)*)"
#octave = r"-2|-1|0|1|2|3|4|5|6|7|8"
#rest = r"R"
#pitch = r"(?P<PITCH>((A|B|C|D|E|F|G)(#+|b+)*)(-2|-1|0|1|2|3|4|5|6|7|8)|(R))"
#tval = r"w|h|q|e|s|t|f"
#dot = r"\.+"
#let = r"t|3|5|7|9"
#tmod = dot + r"|" + let
#tmod = r"(?P<TMOD>" + dot + r"|" + let + r")"
En caso de que haya que concatenar dos variables regex, una seguida de la otra, lo mejor es escribir manualmente la expresión
#duration = r"(?P<DURATION>(w|h|q|e|s|t|f)(\.+|(t|3|5|7|9))*)"
# Regex para identificar los lexemas
# Con 4 claves es suficiente
Mnote = r"(?P<MNOTE>^(((A|B|C|D|E|F|G)(#+|b+)*)(\-2|\-1|0|1|2|3|4|5|6|7|8)|(R))((w|h|q|e|s|t|f)(\.+|(t|3|5|7|9))?))" # A#e
idVozInst = r"(?P<IDVOZINST>^[a-z]+)" #right left letras minusculas
Shell
500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
>>>
Assistant
Warnings
May be ignored if you are happy with your program.
A01411625_EntrenamientoDia22.py
Line 18 : Redefining name 'linea' from outer scope (line 322)
Line 53 : Redefining name 'patron' from outer scope (line 325)
Line 57 : Redefining name 'token' from outer scope (line 335)
Line 111 : Redefining name 'tokens' from outer scope (line 294)
Line 112 : Redefining name 'i' from outer scope (line 333)
Line 116 : Redefining name 'token' from
Escribe aquí para buscar
20°C
ESP
LAA
08:22 p. m.
20/03/2022

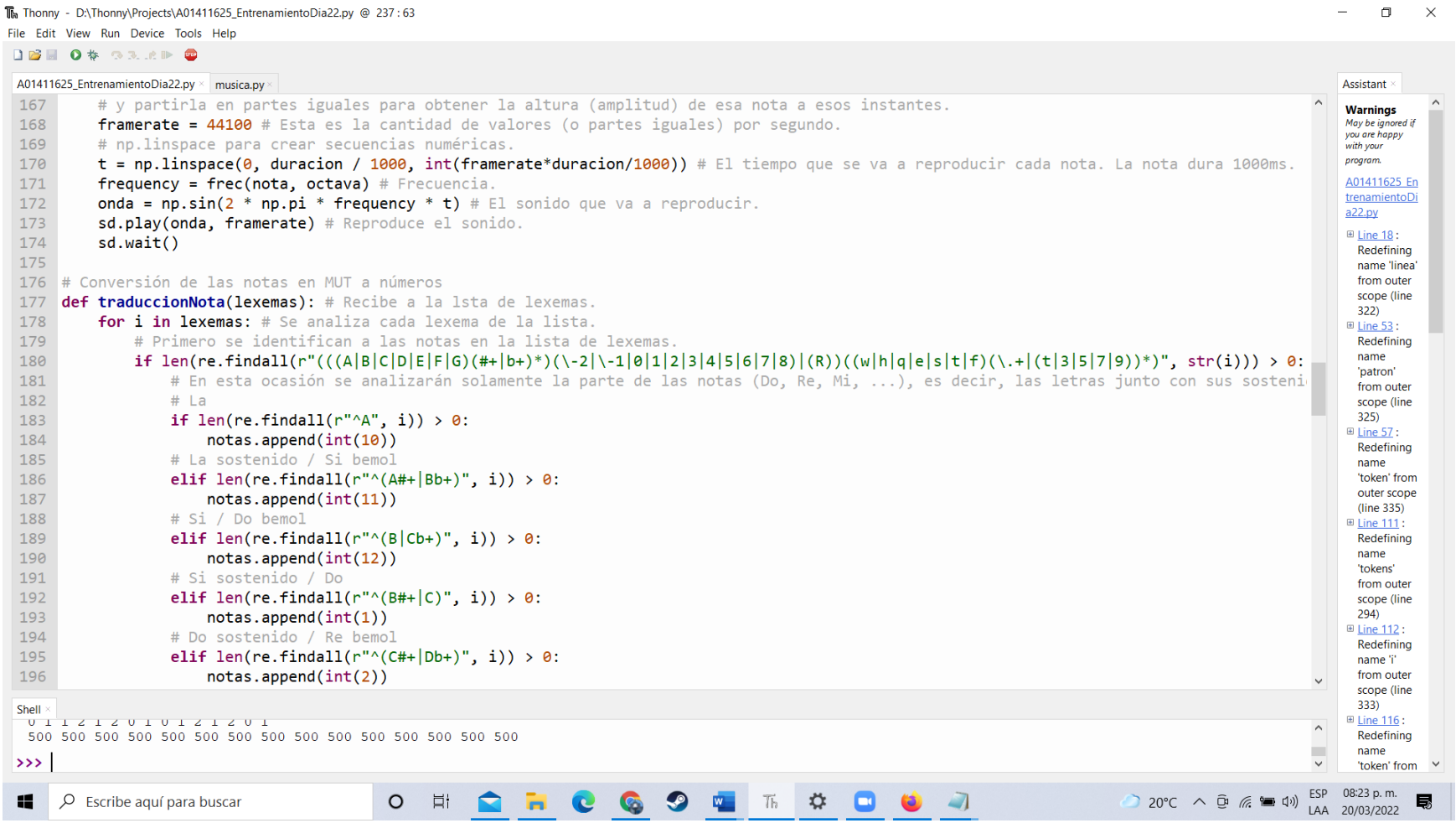

Thonny - D:\Thonny\Projects\A01411625_EntrenamientoDia22.py @ 237 : 63
File Edit View Run Device Tools Help
A01411625_EntrenamientoDia22.py
musica.py
Assistant
Warnings
May be ignored if you are happy with your program.
A01411625_EntrenamientoDia22.py
Line 18 : Redefining name 'linea' from outer scope (line 322)
Line 53 : Redefining name 'patron' from outer scope (line 325)
Line 57 : Redefining name 'token' from outer scope (line 335)
Line 111 : Redefining name 'tokens' from outer scope (line 294)
Line 112 : Redefining name 'i' from outer scope (line 333)
Line 116 : Redefining name 'token' from
# Regex para identificar los lexemas
# Con 4 claves es suficiente
Mnote = r"(?P<MNOTE>^(((A|B|C|D|E|F|G)(#+|b+)*)(\-2|\-1|0|1|2|3|4|5|6|7|8)|(R))((w|h|q|e|s|t|f)(\.+|(t|3|5|7|9))?))" # A#e
idVozInst = r"(?P<IDVOZINST>^[a-z]+)" #right left letras minusculas
py = r"(?P<PY>^\|)" # |
comentarios = r"(?P<COMENTARIOS>^#.+)" # #Cometarios
# Se define el patrón a reconocer tomando en cuenta cualquiera de las 4 claves.
patron = re.compile("|".join([Mnote, idVozInst, py, comentarios]))
grupos = patron.scanner(linea) # Se hace un escaneo o análisis de la linea.
token = grupos.match() # Se identifica si hay una coincidenia
# Si no se reconoce ningún lexema, manda mensaje de error
if str(token) == "None":
print("ERROR: Hay un lexema no válido.\nEl lexema no válido se encuentra justo después de este último que se imprimió.")
raise SystemExit(" Repito - ERROR: Hay un lexema no válido.") # Se imprime un mensaje de error.
print(token.lastgroup, token.group()) # Se imprime la coincidencia junto con su clave (tipo de lexema)
tokenAInsertar = int(0) # Se define la variale para saber cual valor de token insertar.
# El token con valor de 1 representa Mnote
if len(re.findall(Mnote, linea)) > 0: # Si hay al menos una coincidencia...
tokenAInsertar = int(1) # Se asigna el valor del token
coincidencia = re.search(Mnote, linea) # Se busca una coincidencia en específico
if coincidencia.start() == 0: # Si la coincidencia está al principio de la linea...
tokens.append(tokenAInsertar) # Se anexa el token a su lista correspondiente
Shell
500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
>>>
Escribe aquí para buscar
20°C
ESP
LAA
08:22 p. m.
20/03/2022

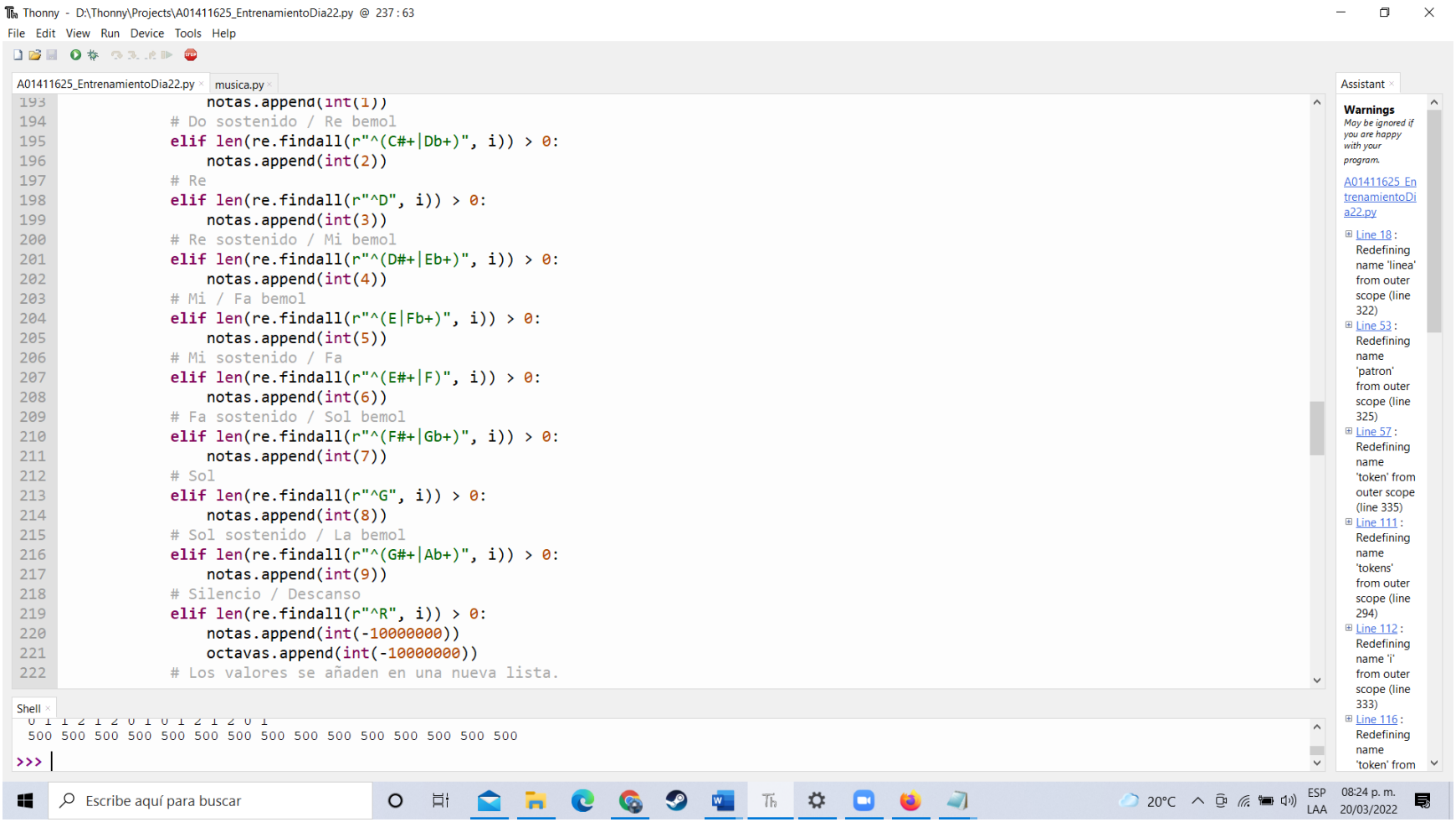

Thonny - D:\Thonny\Projects\A01411625_EntrenamientoDia22.py @ 237 : 63
File Edit View Run Device Tools Help
A01411625_EntrenamientoDia22.py
musica.py
Assistant
Warnings
May be ignored if you are happy with your program.
A01411625_EntrenamientoDia22.py
Line 18 : Redefining name 'linea' from outer scope (line 322)
Line 53 : Redefining name 'patron' from outer scope (line 325)
Line 57 : Redefining name 'token' from outer scope (line 335)
Line 111 : Redefining name 'tokens' from outer scope (line 294)
Line 112 : Redefining name 'i' from outer scope (line 333)
Line 116 : Redefining name 'token' from
70 tokenAInsertar = int(1) # Se asigna el valor del token
71 coincidencia = re.search(Mnote, linea) # Se busca una coincidencia en específico
72 if coincidencia.start() == 0: # Si la coincidencia está al principio de la linea...
73 tokens.append(tokenAInsertar) # Se anexa el token a su lista correspondiente
74
75 # El token con valor de 2 representa el identificador de voz o instrumento
76 elif len(re.findall(idVozInst, linea)) > 0: # Si hay al menos una coincidencia...
77 tokenAInsertar = int(2) # Se asigna el valor del token
78 coincidencia = re.search(idVozInst, linea) # Se busca una coincidencia en específico
79 if coincidencia.start() == 0: # Si la coincidencia está al principio de la linea...
80 tokens.append(tokenAInsertar) # Se anexa el token a su lista correspondiente
81
82 # El token con valor de 3 representa el py (|)
83 elif len(re.findall(py, linea)) > 0: # Si hay al menos una coincidencia...
84 tokenAInsertar = int(3) # Se asigna el valor del token
85 coincidencia = re.search(py, linea) # Se busca una coincidencia en específico
86 if coincidencia.start() == 0: # Si la coincidencia está al principio de la linea...
87 tokens.append(tokenAInsertar) # Se anexa el token a su lista correspondiente
88
89 # Los comentarios no generan token.
90
91 # Si no se reconoce ningún lexema...
92 #else:
93 # print("ERROR: Hay un lexema no válido.")
94 # raise SystemExit("ERROR: Hay un lexema no válido.") # Se imprime un mensaje de error.
95
96 # Cualquier lexema que se reconoza a excepción de los comentarios, se guardan en la lista de lexemas.
97 if len(re.findall(comentarios, linea)) == 0:
98 lexemas.append(token.group())
99
Shell
500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
>>>
Escribe aquí para buscar
20°C
ESP
LAA
08:22 p. m.
20/03/2022

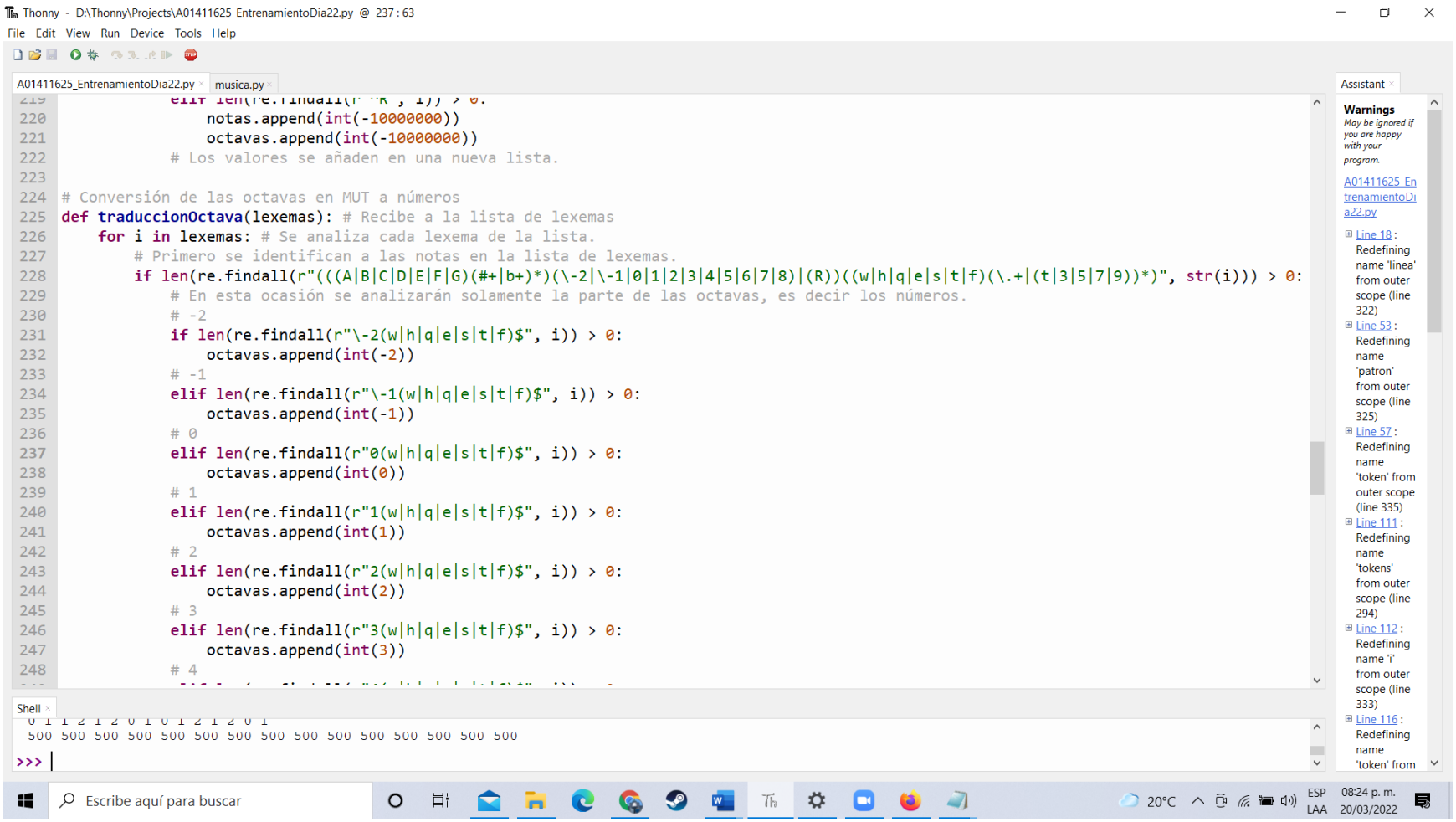

Thonny - D:\Thonny\Projects\A01411625_EntrenamientoDia22.py @ 237 : 63
File Edit View Run Device Tools Help
A01411625_EntrenamientoDia22.py
musica.py
Assistant
Warnings
May be ignored if you are happy with your program.
A01411625_EntrenamientoDia22.py
Line 18 : Redefining name 'linea' from outer scope (line 322)
Line 53 : Redefining name 'patron' from outer scope (line 325)
Line 57 : Redefining name 'token' from outer scope (line 335)
Line 111 : Redefining name 'tokens' from outer scope (line 294)
Line 112 : Redefining name 'i' from outer scope (line 333)
Line 116 : Redefining name 'token' from
95
96 # Cualquier lexema que se reconoza a excepción de los comentarios, se guardan en la lista de lexemas.
97 if len(re.findall(comentarios, linea)) == 0:
98 lexemas.append(token.group())
99
100 # Arreglo: Token(1)
101 # Arreglo: C4e
102
103 # Parte 2: Analizador de Sintaxis
104
105 # Gramática BNF:
106 # <Lineas> ::= idVozInst <Notas> | <Lineas>
107 # <Notas> :: = Mnote (e | py | <Notas>)
108
109
110 # Las claves se usan en la sintaxis para reconocer el orden, si vienen el orden correcto.
111 def Parser(tokens): # Recibe a la lista de tokens
112 for i in tokens: # Analiza cada token de la lista.
113 Lineas(i) # Llama a la función.
114
115 # <Lineas> ::= idVozInst <Notas> | <Lineas>
116 def Lineas(token): # Recibe al token
117 if token == 2: # Si es un identificador de voz o instrumento...
118 sigToken = dameToken() # Pasa al siguiente token
119 if sigToken == 0: # Si no hay más tokens por analizar...
120 return # Termina la función.
121 elif sigToken == 1: # Si es una nota...
122 Notas(sigToken) # Llama a la función.
123
124 # Si no es ninguna de las opciones anteriores marca error
Shell
500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
>>>
Escribe aquí para buscar
20°C
ESP
LAA
08:22 p. m.
20/03/2022

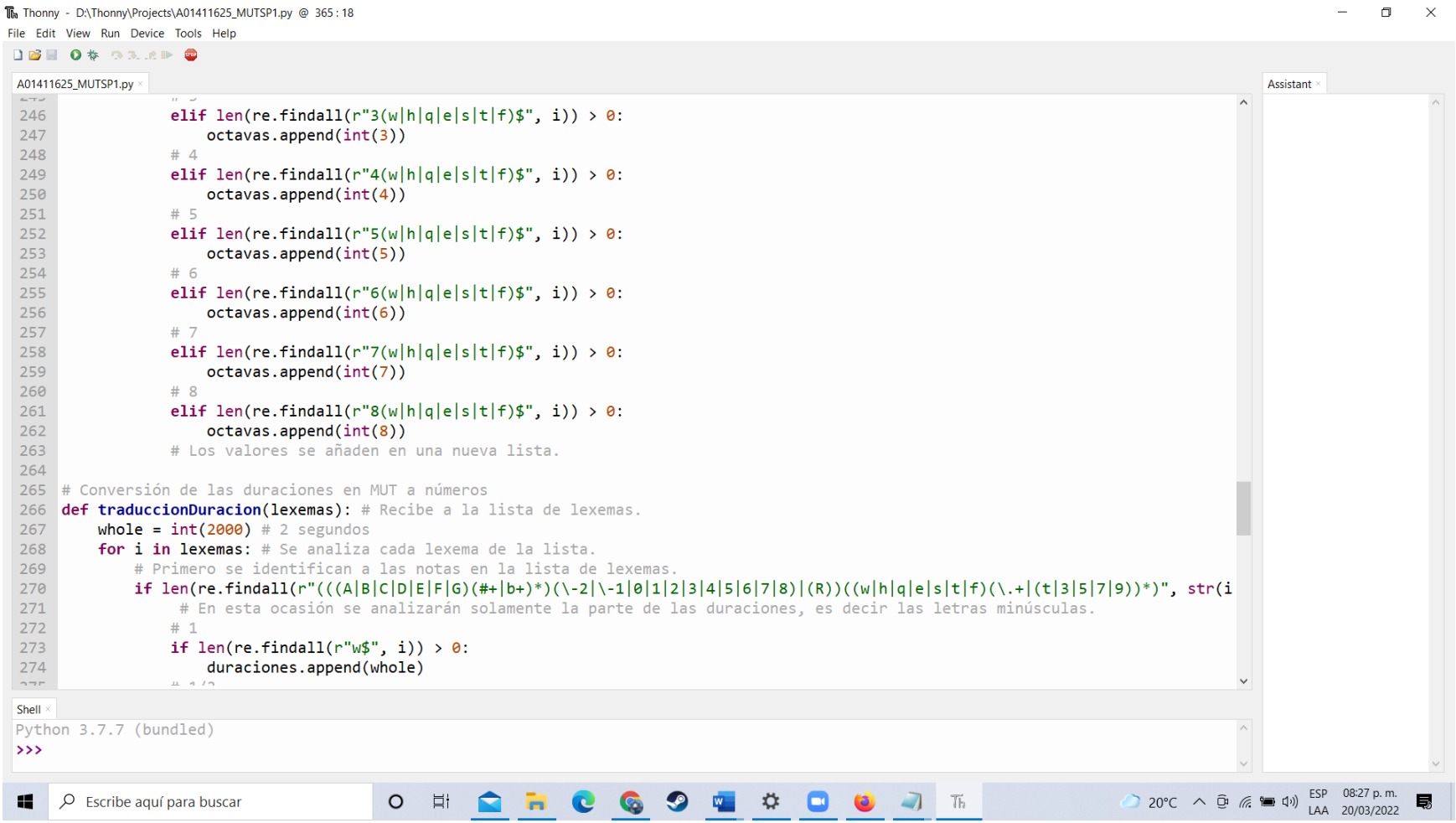

Thonny - D:\Thonny\Projects\A01411625_EntrenamientoDia22.py @ 237 : 63
File Edit View Run Device Tools Help
A01411625_EntrenamientoDia22.py
musica.py
sigToken = dameToken() # Pasa al siguiente token
if sigToken == 0: # Si no hay más tokens por analizar...
return # Termina la función.
elif sigToken == 1: # Si es una nota...
Notas(sigToken) # Llama a la función.
# Si no es ninguna de las opciones anteriores marca error.
else:
print("ERROR: Se esperaba una nota en el texto")
raise SystemExit("Repito - ERROR: Se esperaba una nota")
# Si no empieza con un identificador de voz o instrumento manda mensaje de error.
else:
print("ERROR: Se esperaba un identificacor de Voz o Instrumento al inicio de una linea.")
raise SystemExit("Repito - ERROR: Se esperaba un identificacor de Voz o Instrumento.")
<Notas> :: = Mnote (e | py | <Notas>)
def Notas(token): # Recibe el token
if token == 1: # Si es una nota...
sigToken = dameToken() # Pasa al siguiente token
if sigToken == 0: # Si no hay más tokens por analizar...
return # Termina la función.
Notas(sigToken) # Llama recursivamente a la función de Notas
elif token == 3: # Si es un py (|)...
sigToken = dameToken() # Pasa al siguiente token
if sigToken == 0: # Si no hay más tokens por analizar...
return # Termina la función.
Notas(sigToken) # Llama recursivamente a la función de Notas.
elif token == 2: # Si es un identificador de voz o instrumento...
Lineas(token) # Llama a la función de Líneas.
Assistant
Warnings
May be ignored if you are happy with your program.
A01411625_EntrenamientoDia22.py
Line 18 : Redefining name 'linea' from outer scope (line 322)
Line 53 : Redefining name 'patron' from outer scope (line 325)
Line 57 : Redefining name 'token' from outer scope (line 335)
Line 111 : Redefining name 'tokens' from outer scope (line 294)
Line 112 : Redefining name 'i' from outer scope (line 333)
Line 116 : Redefining name 'token' from
Shell
500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
>>>
Escribe aquí para buscar
20°C
ESP
LAA
08:22 p. m.
20/03/2022

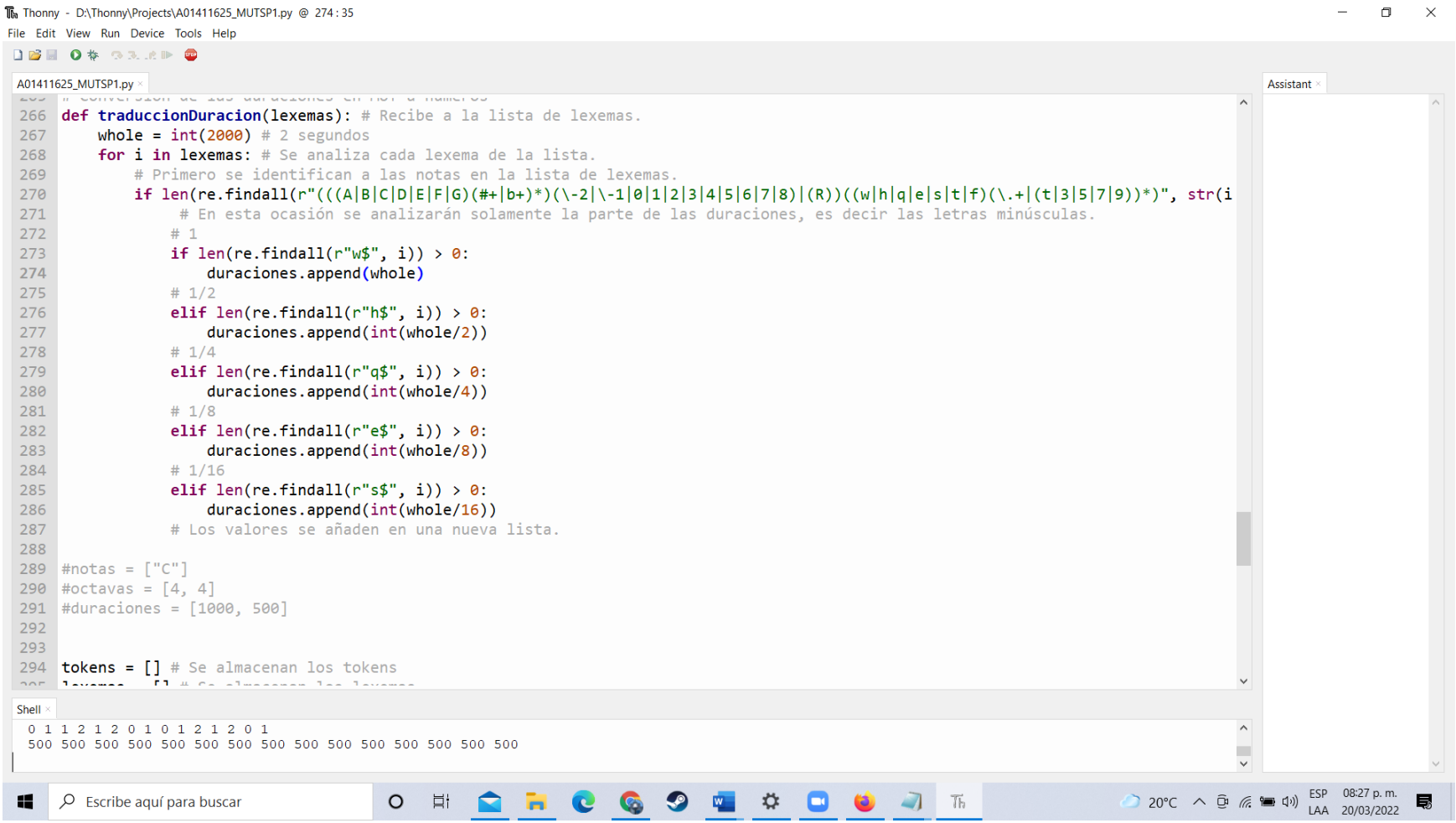

Thonny - D:\Thonny\Projects\A01411625_EntrenamientoDia22.py @ 237 : 63
File Edit View Run Device Tools Help
A01411625_EntrenamientoDia22.py
musica.py
Assistant
Warnings
May be ignored if you are happy with your program.
A01411625_EntrenamientoDia22.py
Line 18 : Redefining name 'linea' from outer scope (line 322)
Line 53 : Redefining name 'patron' from outer scope (line 325)
Line 57 : Redefining name 'token' from outer scope (line 335)
Line 111 : Redefining name 'tokens' from outer scope (line 294)
Line 112 : Redefining name 'i' from outer scope (line 333)
Line 116 : Redefining name 'token' from
return # Termina la función.
Notas(sigToken) # Llama recursivamente a la función de Notas.
elif token == 2: # Si es un identificador de voz o instrumento...
Lineas(token) # Llama a la función de Líneas.
# Si no es ninguna de las opciones anteriores manda mensaje de error.
else:
print("ERROR: Se esperaba una nota en el texto")
raise SystemExit("Repito - ERROR:Se esperaba una nota")
#Parte 3: Ejecutor de Música
La frecuencia son las vibraciones de la nota por segundo.
La frecuencia determina si el sonido será grave o agudo.
def frec(nota: int, octava: int) -> int: # Para calcular la frecuencia se necesita de la nota y la octava.
# Si el resultado de la resta es mayor, suena más agudo porque vibra con una frecuencia más alta la nota.
expo = octava * 12 + (nota - 40) # Exponente
# LA 440 se encuentra por encima del Do central del piano, vibrando a 440 Hz,
# y ha sido universalmente aceptado como el tono según el cual deben afinarse todos los instrumentos para un concierto,
# que es el mismo tono que usa el diapasón tradicional.
return int(440 * ((2 ** (1 / 12)) ** expo)) # Fórmula para conseguir la frecuencia de cada nota.
# Esta función reproduce sonidos tomando en cuenta la nota, octava, y duración.
def play(nota: int, octava: int, duracion: int)->None:
# El framerate consiste en agarrar una onda (la cual tiene cierta cantidad de oscilaciones por segundo)
# y partirla en partes iguales para obtener la altura (amplitud) de esa nota a esos instantes.
framerate = 44100 # Esta es la cantidad de valores (o partes iguales) por segundo.
# np.linspace para crear secuencias numéricas.
t = np.linspace(0, duracion / 1000, int(framerate*duracion/1000)) # El tiempo que se va a reproducir cada nota. La nota dura 1000ms.
frequency = frec(nota, octava) # Frecuencia.
Shell
500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
>>>
Escribe aquí para buscar
20°C
ESP
LAA
08:23 p. m.
20/03/2022

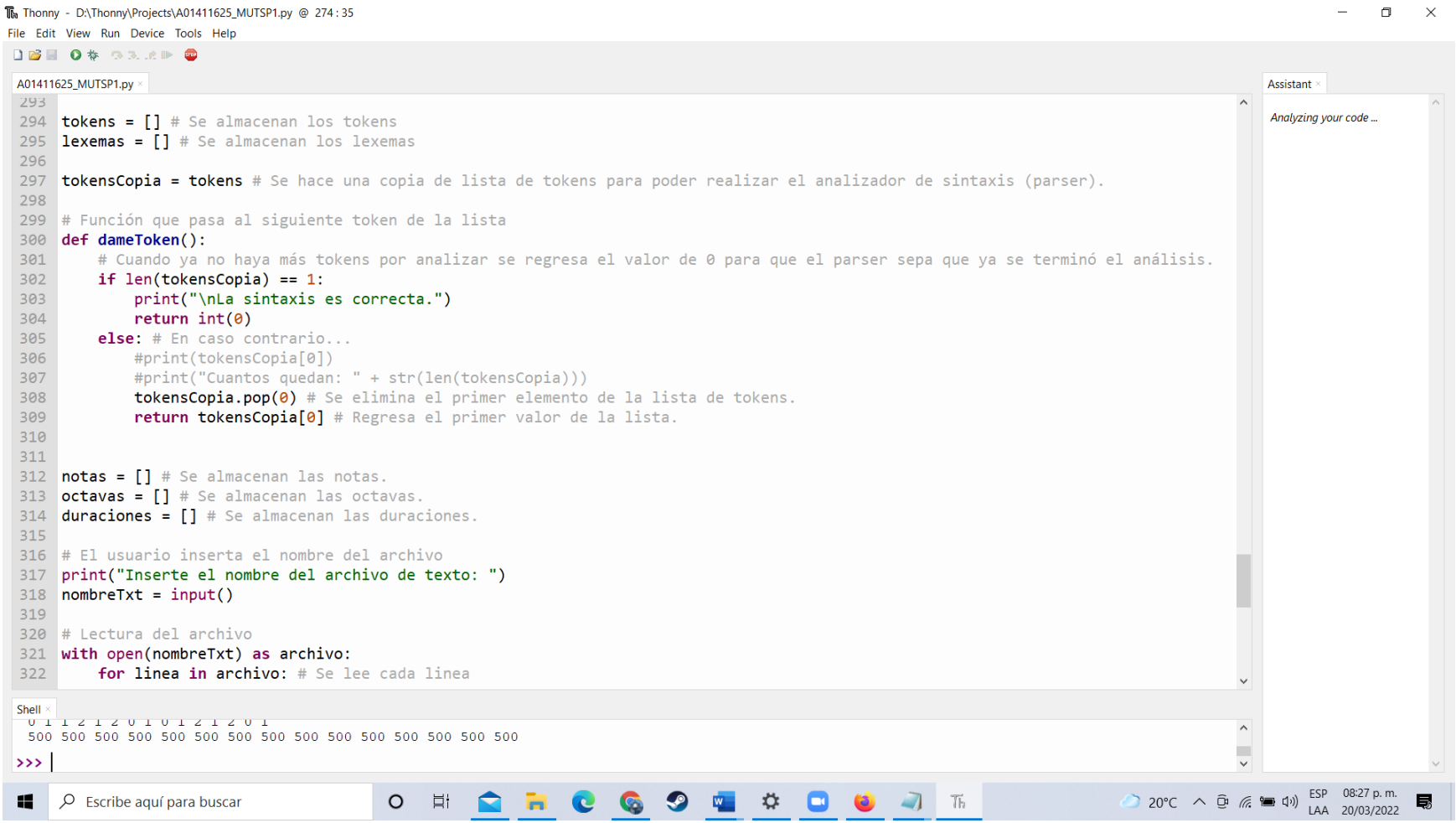

Thonny - D:\Thonny\Projects\A01411625_EntrenamientoDia22.py @ 237 : 63
File Edit View Run Device Tools Help
A01411625_EntrenamientoDia22.py
musica.py
Assistant
Warnings
May be ignored if you are happy with your program.
A01411625_EntrenamientoDia22.py
Line 18 : Redefining name 'linea' from outer scope (line 322)
Line 53 : Redefining name 'patron' from outer scope (line 325)
Line 57 : Redefining name 'token' from outer scope (line 335)
Line 111 : Redefining name 'tokens' from outer scope (line 294)
Line 112 : Redefining name 'i' from outer scope (line 333)
Line 116 : Redefining name 'token' from
# y partirla en partes iguales para obtener la altura (amplitud) de esa nota a esos instantes.
framerate = 44100 # Esta es la cantidad de valores (o partes iguales) por segundo.
# np.linspace para crear secuencias numéricas.
t = np.linspace(0, duracion / 1000, int(framerate*duracion/1000)) # El tiempo que se va a reproducir cada nota. La nota dura 1000ms.
frequency = frec(nota, octava) # Frecuencia.
onda = np.sin(2 * np.pi * frequency * t) # El sonido que va a reproducir.
sd.play(onda, framerate) # Reproduce el sonido.
sd.wait()
# Conversión de las notas en MUT a números
def traduccionNota(lexemas): # Recibe a la lsta de lexemas.
for i in lexemas: # Se analiza cada lexema de la lista.
# Primero se identifican a las notas en la lista de lexemas.
if len(re.findall(r"(((A|B|C|D|E|F|G)(#+|b+)*)(\-2|\-1|0|1|2|3|4|5|6|7|8)|(R))((w|h|q|e|s|t|f)(\.+|(t|3|5|7|9))*)", str(i))) > 0:
# En esta ocasión se analizarán solamente la parte de las notas (Do, Re, Mi, ...), es decir, las letras junto con sus sosteni
# La
if len(re.findall(r"^A", i)) > 0:
notas.append(int(10))
# La sostenido / Si bemol
elif len(re.findall(r"^(A#+|Bb+)", i)) > 0:
notas.append(int(11))
# Si / Do bemol
elif len(re.findall(r"^(B|Cb+)", i)) > 0:
notas.append(int(12))
# Si sostenido / Do
elif len(re.findall(r"^(B#+|C)", i)) > 0:
notas.append(int(1))
# Do sostenido / Re bemol
elif len(re.findall(r"^(C#+|Db+)", i)) > 0:
notas.append(int(2))
Shell
500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
>>>
Escribe aquí para buscar
20°C
ESP
LAA
08:23 p. m.
20/03/2022

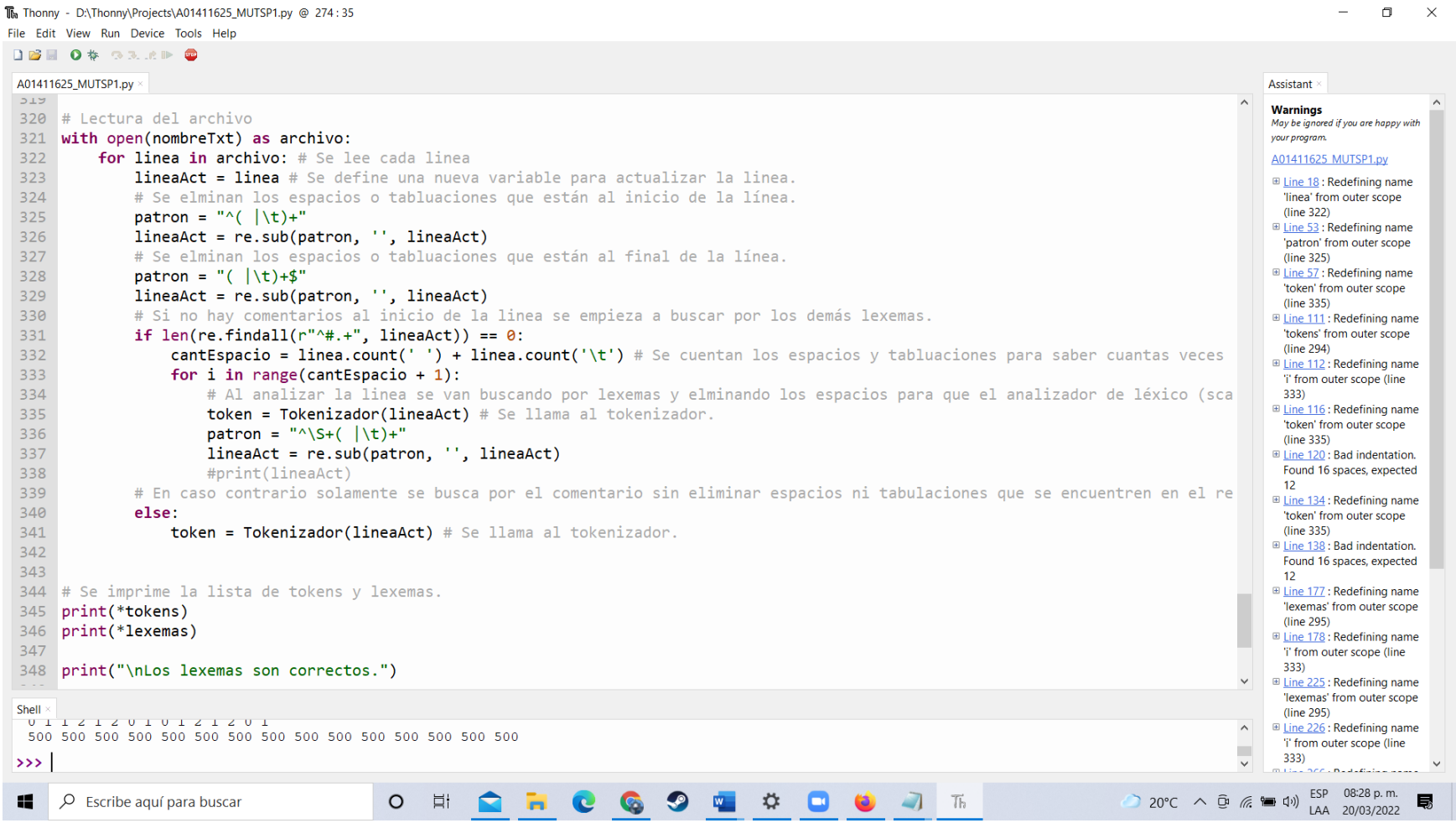

Thonny - D:\Thonny\Projects\A01411625_EntrenamientoDia22.py @ 237 : 63
File Edit View Run Device Tools Help
A01411625_EntrenamientoDia22.py
musica.py
Assistant
Warnings
May be ignored if you are happy with your program.
A01411625_EntrenamientoDia22.py
Line 18 : Redefining name 'linea' from outer scope (line 322)
Line 53 : Redefining name 'patron' from outer scope (line 325)
Line 57 : Redefining name 'token' from outer scope (line 335)
Line 111 : Redefining name 'tokens' from outer scope (line 294)
Line 112 : Redefining name 'i' from outer scope (line 333)
Line 116 : Redefining name 'token' from
notas.append(int(1))
# Do sostenido / Re bemol
elif len(re.findall(r"^(C#+|Db+)", i)) > 0:
notas.append(int(2))
# Re
elif len(re.findall(r"^D", i)) > 0:
notas.append(int(3))
# Re sostenido / Mi bemol
elif len(re.findall(r"^(D#+|Eb+)", i)) > 0:
notas.append(int(4))
# Mi / Fa bemol
elif len(re.findall(r"^(E|Fb+)", i)) > 0:
notas.append(int(5))
# Mi sostenido / Fa
elif len(re.findall(r"^(E#+|F)", i)) > 0:
notas.append(int(6))
# Fa sostenido / Sol bemol
elif len(re.findall(r"^(F#+|Gb+)", i)) > 0:
notas.append(int(7))
# Sol
elif len(re.findall(r"^G", i)) > 0:
notas.append(int(8))
# Sol sostenido / La bemol
elif len(re.findall(r"^(G#+|Ab+)", i)) > 0:
notas.append(int(9))
# Silencio / Descanso
elif len(re.findall(r"^R", i)) > 0:
notas.append(int(-10000000))
octavas.append(int(-10000000))
# Los valores se añaden en una nueva lista.
Shell
500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
>>>
Escribe aquí para buscar
20°C
ESP
LAA
08:24 p. m.
20/03/2022

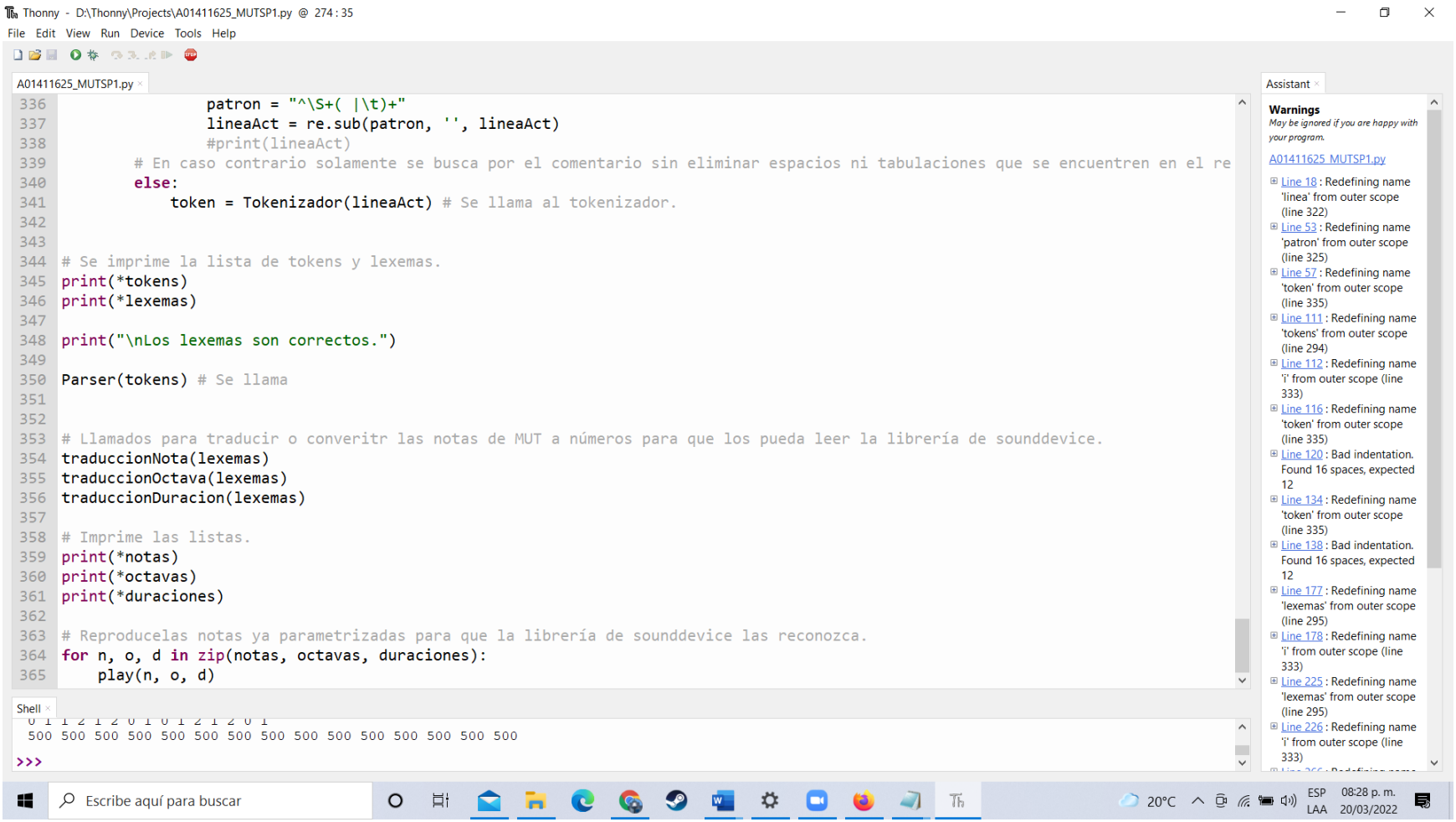

Thonny - D:\Thonny\Projects\A01411625_EntrenamientoDia22.py @ 237 : 63
File Edit View Run Device Tools Help
A01411625_EntrenamientoDia22.py
musica.py
Assistant
Warnings
May be ignored if you are happy with your program.
A01411625_EntrenamientoDia22.py
Line 18 : Redefining name 'linea' from outer scope (line 322)
Line 53 : Redefining name 'patron' from outer scope (line 325)
Line 57 : Redefining name 'token' from outer scope (line 335)
Line 111 : Redefining name 'tokens' from outer scope (line 294)
Line 112 : Redefining name 'i' from outer scope (line 333)
Line 116 : Redefining name 'token' from
notas.append(int(-10000000))
octavas.append(int(-10000000))
# Los valores se añaden en una nueva lista.
# Conversión de las octavas en MUT a números
def traduccionOctava(lexemas): # Recibe a la lista de lexemas
for i in lexemas: # Se analiza cada lexema de la lista.
# Primero se identifican a las notas en la lista de lexemas.
if len(re.findall(r"(((A|B|C|D|E|F|G)(#+|b+)*)(\-2|\-1|0|1|2|3|4|5|6|7|8)|(R))((w|h|q|e|s|t|f)(\.+|(t|3|5|7|9))*)", str(i))) > 0:
# En esta ocasión se analizarán solamente la parte de las octavas, es decir los números.
# -2
if len(re.findall(r"\-2(w|h|q|e|s|t|f)$", i)) > 0:
octavas.append(int(-2))
# -1
elif len(re.findall(r"\-1(w|h|q|e|s|t|f)$", i)) > 0:
octavas.append(int(-1))
# 0
elif len(re.findall(r"0(w|h|q|e|s|t|f)$", i)) > 0:
octavas.append(int(0))
# 1
elif len(re.findall(r"1(w|h|q|e|s|t|f)$", i)) > 0:
octavas.append(int(1))
# 2
elif len(re.findall(r"2(w|h|q|e|s|t|f)$", i)) > 0:
octavas.append(int(2))
# 3
elif len(re.findall(r"3(w|h|q|e|s|t|f)$", i)) > 0:
octavas.append(int(3))
# 4
Shell
500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
>>>
Escribe aquí para buscar
20°C
ESP
LAA
08:24 p. m.
20/03/2022


Thonny - D:\Thonny\Projects\A01411625_MUTSP1.py @ 365 : 18
File Edit View Run Device Tools Help
A01411625_MUTSP1.py
Assistant
elif len(re.findall(r"3(w|h|q|e|s|t|f)$", i)) > 0:
octavas.append(int(3))
# 4
elif len(re.findall(r"4(w|h|q|e|s|t|f)$", i)) > 0:
octavas.append(int(4))
# 5
elif len(re.findall(r"5(w|h|q|e|s|t|f)$", i)) > 0:
octavas.append(int(5))
# 6
elif len(re.findall(r"6(w|h|q|e|s|t|f)$", i)) > 0:
octavas.append(int(6))
# 7
elif len(re.findall(r"7(w|h|q|e|s|t|f)$", i)) > 0:
octavas.append(int(7))
# 8
elif len(re.findall(r"8(w|h|q|e|s|t|f)$", i)) > 0:
octavas.append(int(8))
# Los valores se añaden en una nueva lista.
# Conversión de las duraciones en MUT a números
def traduccionDuracion(lexemas): # Recibe a la lista de lexemas.
whole = int(2000) # 2 segundos
for i in lexemas: # Se analiza cada lexema de la lista.
Primero se identifican a las notas en la lista de lexemas.
if len(re.findall(r"(((A|B|C|D|E|F|G)(#+|b+)*)(\-2|\-1|0|1|2|3|4|5|6|7|8)|(R))((w|h|q|e|s|t|f)(\.+|(t|3|5|7|9))*)", str(i
# En esta ocasión se analizarán solamente la parte de las duraciones, es decir las letras minúsculas.
# 1
if len(re.findall(r"w$", i)) > 0:
duraciones.append(whole)
Shell
Python 3.7.7 (bundled)
>>>
Escribe aquí para buscar
20°C
ESP
LAA
08:27 p. m.
20/03/2022


Thonny - D:\Thonny\Projects\A01411625_MUTSP1.py @ 274 : 35
File Edit View Run Device Tools Help
A01411625_MUTSP1.py
Assistant
def traduccionDuracion(lexemas): # Recibe a la lista de lexemas.
whole = int(2000) # 2 segundos
for i in lexemas: # Se analiza cada lexema de la lista.
Primero se identifican a las notas en la lista de lexemas.
if len(re.findall(r"(((A|B|C|D|E|F|G)(#+|b+)*)(\-2|\-1|0|1|2|3|4|5|6|7|8)|(R))((w|h|q|e|s|t|f)(\.+|(t|3|5|7|9))*)", str(i
# En esta ocasión se analizarán solamente la parte de las duraciones, es decir las letras minúsculas.
# 1
if len(re.findall(r"w$", i)) > 0:
duraciones.append(whole)
# 1/2
elif len(re.findall(r"h$", i)) > 0:
duraciones.append(int(whole/2))
# 1/4
elif len(re.findall(r"q$", i)) > 0:
duraciones.append(int(whole/4))
# 1/8
elif len(re.findall(r"e$", i)) > 0:
duraciones.append(int(whole/8))
# 1/16
elif len(re.findall(r"s$", i)) > 0:
duraciones.append(int(whole/16))
# Los valores se añaden en una nueva lista.
#notas = ["C"]
#octavas = [4, 4]
#duraciones = [1000, 500]
tokens = [] # Se almacenan los tokens
Shell
0 1 1 2 1 2 0 1 0 1 2 1 2 0 1
500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
Escribe aquí para buscar
20°C
ESP
LAA
08:27 p. m.
20/03/2022


Thonny - D:\Thonny\Projects\A01411625_MUTSP1.py @ 274 : 35
File Edit View Run Device Tools Help
A01411625_MUTSP1.py
Assistant
Analyzing your code ...
293
294 tokens = [] # Se almacenan los tokens
295 lexemas = [] # Se almacenan los lexemas
296
297 tokensCopia = tokens # Se hace una copia de lista de tokens para poder realizar el analizador de sintaxis (parser).
298
299 # Función que pasa al siguiente token de la lista
300 def dameToken():
301 # Cuando ya no haya más tokens por analizar se regresa el valor de 0 para que el parser sepa que ya se terminó el análisis.
302 if len(tokensCopia) == 1:
303 print("\nLa sintaxis es correcta.")
304 return int(0)
305 else: # En caso contrario...
306 #print(tokensCopia[0])
307 #print("Cuantos quedan: " + str(len(tokensCopia)))
308 tokensCopia.pop(0) # Se elimina el primer elemento de la lista de tokens.
309 return tokensCopia[0] # Regresa el primer valor de la lista.
310
311
312 notas = [] # Se almacenan las notas.
313 octavas = [] # Se almacenan las octavas.
314 duraciones = [] # Se almacenan las duraciones.
315
316 # El usuario inserta el nombre del archivo
317 print("Inserte el nombre del archivo de texto: ")
318 nombreTxt = input()
319
320 # Lectura del archivo
321 with open(nombreTxt) as archivo:
322 for linea in archivo: # Se lee cada linea
Shell
0 1 1 2 1 2 0 1 0 1 2 1 2 0 1
500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
>>>
Escribe aquí para buscar
20°C
ESP
LAA
08:27 p. m.
20/03/2022


Thonny - D:\Thonny\Projects\A01411625_MUTSP1.py @ 274 : 35
File Edit View Run Device Tools Help
A01411625_MUTSP1.py
Lectura del archivo
with open(nombreTxt) as archivo:
for linea in archivo: # Se lee cada linea
lineaAct = linea # Se define una nueva variable para actualizar la linea.
# Se elminan los espacios o tabluaciones que están al inicio de la línea.
patron = "^(|\t)+"
lineaAct = re.sub(patron, '', lineaAct)
# Se elminan los espacios o tabluaciones que están al final de la línea.
patron = "(|\t)+$"
lineaAct = re.sub(patron, '', lineaAct)
# Si no hay comentarios al inicio de la linea se empieza a buscar por los demás lexemas.
if len(re.findall(r"^#.+", lineaAct)) == 0:
cantEspacio = linea.count(' ') + linea.count('\t') # Se cuentan los espacios y tabluaciones para saber cuantas veces
for i in range(cantEspacio + 1):
# Al analizar la linea se van buscando por lexemas y elminando los espacios para que el analizador de léxico (sca
token = Tokenizador(lineaAct) # Se llama al tokenizador.
patron = "^\S+(|\t)+"
lineaAct = re.sub(patron, '', lineaAct)
#print(lineaAct)
# En caso contrario solamente se busca por el comentario sin eliminar espacios ni tabulaciones que se encuentren en el re
else:
token = Tokenizador(lineaAct) # Se llama al tokenizador.
# Se imprime la lista de tokens y lexemas.
print(*tokens)
print(*lexemas)
print("\nLos lexemas son correctos.")
Shell
500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
>>>
Assistant
Warnings
May be ignored if you are happy with your program.
A01411625_MUTSP1.py
Line 18 : Redefining name 'linea' from outer scope (line 322)
Line 53 : Redefining name 'patron' from outer scope (line 325)
Line 57 : Redefining name 'token' from outer scope (line 335)
Line 111 : Redefining name 'tokens' from outer scope (line 294)
Line 112 : Redefining name 'i' from outer scope (line 333)
Line 116 : Redefining name 'token' from outer scope (line 335)
Line 120 : Bad indentation. Found 16 spaces, expected 12
Line 134 : Redefining name 'token' from outer scope (line 335)
Line 138 : Bad indentation. Found 16 spaces, expected 12
Line 177 : Redefining name 'lexemas' from outer scope (line 295)
Line 178 : Redefining name 'i' from outer scope (line 333)
Line 225 : Redefining name 'lexemas' from outer scope (line 295)
Line 226 : Redefining name 'i' from outer scope (line 333)
Escribe aquí para buscar
20°C
ESP
LAA
08:28 p. m.
20/03/2022



Situación Problema 1: “No sólo de los lenguajes populares vive un ITC”: aprendiendo ágilmente un pequeño lenguaje

Juan Daniel Rodríguez Oropeza A01411625

20 de Marzo de 2022

```
import re # Librería para usar expresiones regulares
import numpy as np # Librería para usar elementos matemáticos un poco más complejos.
# Librería que funciona cuando al darle un input matemático de una onda sonora, la reproducirá tranformándola en una onda acústica.
import sounddevice as sd
#import sys

# 1. Análisis de Léxico (entrada: texto; salida: lexemas y tokens)
# 2. Análisis de Sintaxis (entrada: lista de tokens; salida: nada (todo bien) error (avisa que hay error de sintaxis))
# 3. Ejecutor de Música (entrada: lista de lexemas de notas, se deslgosa y parametriza; salida: sonido)


# Parte 1: Analizador de léxico
# Tokenizar, converitr tu codigo de entrada en claves
def Tokenizador(linea): # Recibe a la linea
    #sharp = r"(#+)"
    #flat = r"(b+)"

    #accidental = sharp + r"|" + flat

    #letter = r"(A|B|C|D|E|F|G)"

    # Los identificadores en Regex no pueden estar más de una vez.
```

```
#notename = r"(?P<NOTENAME>(A|B|C|D|E|F|G)(#+|b+)*)"

#octave = r"-2|-1|0|1|2|3|4|5|6|7|8"
#rest = r"R"

#pitch = r"(?P<PITCH>((A|B|C|D|E|F|G)(#+|b+)*)(-2|-1|0|1|2|3|4|5|6|7|8)|(R))"

#tval = r"w|h|q|e|s|t|f"

#dot = r"\.+"
#let = r"t|3|5|7|9"

#tmod = dot + r"|" + let
#tmod = r"(?P<TMOD>" + dot + r"|" + let + r")"

# En caso de que haya que concatenar dos variables regex, una seguida de la otra, lo mejor es escribir manualmente la expresión
#duration = r"(?P<DURATION>(w|h|q|e|s|t|f)(\.+|(t|3|5|7|9))*)"

# Regex para identificar los lexemas
# Con 4 claves es suficiente
Mnote = r"(?P<MNOTE>^(((A|B|C|D|E|F|G)(#+|b+)*)(\-2|\-1|0|1|2|3|4|5|6|7|8)|(R))((w|h|q|e|s|t|f)(\.+|(t|3|5|7|9))?))" # A#e
idVozInst = r"(?P<IDVOZINST>^[a-z]+)"  #right left letras minusculas
py = r"(?P<PY>^\|)" # |
```

```
comentarios = r"(?P<COMENTARIOS>^#.+)" # #Comentarios

# Se define el patrón a reconocer tomando en cuenta cualquiera de las 4 claves.
patron = re.compile("|".join([Mnote, idVozInst, py, comentarios]))

grupos = patron.scanner(linea) # Se hace un escaneo o análisis de la linea.

token = grupos.match() # Se identifica si hay una coincidencia

# Si no se reconoce ningún lexema, manda mensaje de error
if str(token) == "None":
    print("ERROR: Hay un lexema no válido.\nEl lexema no válido se encuentra justo después de este último que se imprimió.")
    raise SystemExit(" Repito - ERROR: Hay un lexema no válido.") # Se imprime un mensaje de error.

print(token.lastgroup, token.group()) # Se imprime la coincidencia junto con su clave (tipo de lexema)

tokenAInsertar = int(0) # Se define esta variale para saber cual valor de token insertar.

# El token con valor de 1 representa Mnote
if len(re.findall(Mnote, linea)) > 0: # Si hay al menos una coincidencia...
    tokenAInsertar = int(1) # Se asigna el valor del token
    coincidencia = re.search(Mnote, linea) # Se busca la primera coincidencia que encuentre
    if coincidencia.start() == 0: # Si la coincidencia está al principio de la linea...
```

```
            tokens.append(tokenAInsertar) # Se anexa el token a su lista correspondiente

    # El token con valor de 2 representa el identificador de voz o instrumento
    elif len(re.findall(idVozInst, linea)) > 0: # Si hay al menos una coincidencia...
        tokenAInsertar = int(2) # Se asigna el valor del token
        coincidencia = re.search(idVozInst, linea) # Se busca la primera coincidencia que encuentre
        if coincidencia.start() == 0: # Si la coincidencia está al principio de la linea...
            tokens.append(tokenAInsertar) # Se anexa el token a su lista correspondiente

    # El token con valor de 3 representa el py ( | )
    elif len(re.findall(py, linea)) > 0: # Si hay al menos una coincidencia...
        tokenAInsertar = int(3) # Se asigna el valor del token
        coincidencia = re.search(py, linea) # Se busca la primera coincidencia que encuentre
        if coincidencia.start() == 0: # Si la coincidencia está al principio de la linea...
            tokens.append(tokenAInsertar) # Se anexa el token a su lista correspondiente

    # Los comentarios no generan token.

    # Cualquier lexema que se reconoza a excepción de los comentarios, se guardan en la lista de lexemas.
    if len(re.findall(comentarios, linea)) == 0:
        lexemas.append(token.group())

# Arreglo: Token(1)
```

```
# Arreglo: C4e

# Parte 2: Analizador de Sintaxis

# Gramática BNF:
# <Lineas> ::= idVozInst <Notas> | <Lineas>
# <Notas> :: = Mnote (e | py | <Notas>)



# Las claves se usan en la sintaxis para reconocer el orden, si vienen el orden correcto.
def Parser(tokens): # Recibe a la lista de tokens
    for i in tokens: # Analiza cada token de la lista.
        Lineas(i) # Llama a la función.

# <Lineas> ::= idVozInst <Notas> | <Lineas>
def Lineas(token): # Recibe al token
    if token == 2: # Si es un identificador de voz o instrumento...
        sigToken = dameToken() # Pasa al siguiente token
        if sigToken == 0: # Si no hay más tokens por analizar...
                return # Termina la función.
        elif sigToken == 1: # Si es una nota...
            Notas(sigToken) # Llama a la función.
```

```
        # Si no es ninguna de las opciones anteriores marca error.
        else:
            print("ERROR: Se esperaba una nota en el texto")
            raise SystemExit("Repito - ERROR: Se esperaba una nota")
    # Si no empieza con un identificador de voz o instrumento manda mensaje de error.
    else:
        print("ERROR: Se esperaba un identificacor de Voz o Instrumento al inicio de una linea.")
        raise SystemExit("Repito - ERROR: Se esperaba un identificacor de Voz o Instrumento.")

# <Notas> :: = Mnote (e | py | <Notas>)
def Notas(token): # Recibe el token
    if token == 1: # Si es una nota...
        sigToken = dameToken() # Pasa al siguiente token
        if sigToken == 0: # Si no hay más tokens por analizar...
                return # Termina la función.
        Notas(sigToken) # Llama recursivamente a la función de Notas
    elif token == 3: # Si es un py ( | )...
        sigToken = dameToken() # Pasa al siguiente token
        if sigToken == 0: # Si no hay más tokens por analizar...
            return # Termina la función.
        Notas(sigToken) # Llama recursivamente a la función de Notas.
    elif token == 2: # Si es un identificador de voz o instrumento...
        Lineas(token) # Llama a la función de Líneas.
```

```
    # Si no es ninguna de las opciones anteriores manda mensaje de error.
    else:
        print("ERROR: Se esperaba una nota en el texto")
        raise SystemExit("Repito - ERROR:Se esperaba una nota")

#Parte 3: Ejecutor de Música

# La frecuencia son las vibraciones de la nota por segundo.
# La frecuencia determina si el sonido será grave o agudo.
def frec(nota: int, octava: int) -> int: # Para calcular la frecuencia se necesita de la nota y la octava.
    # Si el resultado de la resta es mayor, suena más agudo porque vibra con una frecuencia más alta la nota.
    expo = octava * 12 + (nota - 40) # Exponente
    # LA 440 se encuentra por encima del Do central del piano, vibrando a 440 Hz,
    # y ha sido universalmente aceptado como el tono según el cual deben afinarse todos los instrumentos para un concierto,
    # que es el mismo tono que usa el diapasón tradicional.
    return int(440 * ((2 ** (1 / 12)) ** expo)) # Fórmula para conseguir la frecuencia de cada nota.

# Esta función reproduce sonidos tomando en cuenta la nota, octava, y duración.
def play(nota: int, octava: int, duracion: int)->None:
    # El framerate consiste en agarrar una onda (la cual tiene cierta cantidad de oscilaciones por segundo)
    # y partirla en partes iguales para obtener la altura (amplitud) de esa nota a esos instantes.
    framerate = 44100 # Esta es la cantidad de valores (o partes iguales) por segundo.
    # np.linspace para crear secuencias numéricas.
```

```
    t = np.linspace(0, duracion / 1000, int(framerate*duracion/1000)) # El tiempo que se va a reproducir cada nota. La nota dura 1000ms.
    frequency = frec(nota, octava) # Frecuencia.
    onda = np.sin(2 * np.pi * frequency * t) # El sonido que va a reproducir.
    sd.play(onda, framerate) # Reproduce el sonido.
    sd.wait()

# Conversión de las notas en MUT a números
def traduccionNota(lexemas): # Recibe a la lista de lexemas.
    for i in lexemas: # Se analiza cada lexema de la lista.
        # Primero se identifican a las notas en la lista de lexemas.
        if len(re.findall(r"(((A|B|C|D|E|F|G)(#+|b+)*)(\-2|\-1|0|1|2|3|4|5|6|7|8)|(R))((w|h|q|e|s|t|f)(\.+|(t|3|5|7|9))*)", str(i))) > 0:
            # En esta ocasión se analizarán solamente la parte de las notas (Do, Re, Mi, ...), es decir, las letras junto con sus sostenidos y bemoles.
            # La
            if len(re.findall(r"^A", i)) > 0:
                notas.append(int(10))
            # La sostenido / Si bemol
            elif len(re.findall(r"^(A#+|Bb+)", i)) > 0:
                notas.append(int(11))
            # Si / Do bemol
            elif len(re.findall(r"^(B|Cb+)", i)) > 0:
                notas.append(int(12))
            # Si sostenido / Do
            elif len(re.findall(r"^(B#+|C)", i)) > 0:
```

```
    notas.append(int(1))
# Do sostenido / Re bemol
elif len(re.findall(r"^(C#+|Db+)", i)) > 0:
    notas.append(int(2))
# Re
elif len(re.findall(r"^D", i)) > 0:
    notas.append(int(3))
# Re sostenido / Mi bemol
elif len(re.findall(r"^(D#+|Eb+)", i)) > 0:
    notas.append(int(4))
# Mi / Fa bemol
elif len(re.findall(r"^(E|Fb+)", i)) > 0:
    notas.append(int(5))
# Mi sostenido / Fa
elif len(re.findall(r"^(E#+|F)", i)) > 0:
    notas.append(int(6))
# Fa sostenido / Sol bemol
elif len(re.findall(r"^(F#+|Gb+)", i)) > 0:
    notas.append(int(7))
# Sol
elif len(re.findall(r"^G", i)) > 0:
    notas.append(int(8))
# Sol sostenido / La bemol
```

```
        elif len(re.findall(r"^(G#+|Ab+)", i)) > 0:
            notas.append(int(9))
        # Silencio / Descanso
        elif len(re.findall(r"^R", i)) > 0:
            notas.append(int(-10000000))
            octavas.append(int(-10000000))
        # Los valores se añaden en una nueva lista.

# Conversión de las octavas en MUT a números
def traduccionOctava(lexemas): # Recibe a la lista de lexemas
    for i in lexemas: # Se analiza cada lexema de la lista.
        # Primero se identifican a las notas en la lista de lexemas.
        if len(re.findall(r"(((A|B|C|D|E|F|G)(#+|b+)*)(\-2|\-1|0|1|2|3|4|5|6|7|8)|(R))((w|h|q|e|s|t|f)(\.+|(t|3|5|7|9))*)", str(i))) > 0:
            # En esta ocasión se analizarán solamente la parte de las octavas, es decir los números.
            # -2
            if len(re.findall(r"\-2(w|h|q|e|s|t|f)$", i)) > 0:
                octavas.append(int(-2))
            # -1
            elif len(re.findall(r"\-1(w|h|q|e|s|t|f)$", i)) > 0:
                octavas.append(int(-1))
            # 0
            elif len(re.findall(r"0(w|h|q|e|s|t|f)$", i)) > 0:
                octavas.append(int(0))
```

```
# 1
elif len(re.findall(r"1(w|h|q|e|s|t|f)$", i)) > 0:
    octavas.append(int(1))
# 2
elif len(re.findall(r"2(w|h|q|e|s|t|f)$", i)) > 0:
    octavas.append(int(2))
# 3
elif len(re.findall(r"3(w|h|q|e|s|t|f)$", i)) > 0:
    octavas.append(int(3))
# 4
elif len(re.findall(r"4(w|h|q|e|s|t|f)$", i)) > 0:
    octavas.append(int(4))
# 5
elif len(re.findall(r"5(w|h|q|e|s|t|f)$", i)) > 0:
    octavas.append(int(5))
# 6
elif len(re.findall(r"6(w|h|q|e|s|t|f)$", i)) > 0:
    octavas.append(int(6))
# 7
elif len(re.findall(r"7(w|h|q|e|s|t|f)$", i)) > 0:
    octavas.append(int(7))
# 8
elif len(re.findall(r"8(w|h|q|e|s|t|f)$", i)) > 0:
```

```
            octavas.append(int(8))
        # Los valores se añaden en una nueva lista.

# Conversión de las duraciones en MUT a números
def traduccionDuracion(lexemas): # Recibe a la lista de lexemas.
    whole = int(2000) # 2 segundos
    for i in lexemas: # Se analiza cada lexema de la lista.
        # Primero se identifican a las notas en la lista de lexemas.
        if len(re.findall(r"(((A|B|C|D|E|F|G)(#+|b+)*)(\-2|\-1|0|1|2|3|4|5|6|7|8)|(R))((w|h|q|e|s|t|f)(\.+|(t|3|5|7|9))*)", str(i))) > 0:
            # En esta ocasión se analizarán solamente la parte de las duraciones, es decir las letras minúsculas.
            # 1
            if len(re.findall(r"w$", i)) > 0:
                duraciones.append(whole)
            # 1/2
            elif len(re.findall(r"h$", i)) > 0:
                duraciones.append(int(whole/2))
            # 1/4
            elif len(re.findall(r"q$", i)) > 0:
                duraciones.append(int(whole/4))
            # 1/8
            elif len(re.findall(r"e$", i)) > 0:
                duraciones.append(int(whole/8))
            # 1/16
```

```
        elif len(re.findall(r"s$", i)) > 0:
            duraciones.append(int(whole/16))
        # Los valores se añaden en una nueva lista.

#notas = ["C"]
#octavas = [4, 4]
#duraciones = [1000, 500]



tokens = [] # Se almacenan los tokens
lexemas = [] # Se almacenan los lexemas

tokensCopia = tokens # Se hace una copia de lista de tokens para poder realizar el analizador de sintaxis (parser).

# Función que pasa al siguiente token de la lista
def dameToken():
    # Cuando ya no haya más tokens por analizar se regresa el valor de 0 para que el parser sepa que ya se terminó el análisis.
    if len(tokensCopia) == 1:
        print("\nLa sintaxis es correcta.")
        return int(0)
    else: # En caso contrario...
        print(tokensCopia[0])
        #print("Cuantos quedan: " + str(len(tokensCopia)))
```

```
        tokensCopia.pop(0) # Se elimina el primer elemento de la lista de tokens.
        return tokensCopia[0] # Regresa el primer valor de la lista.



notas = [] # Se almacenan las notas.
octavas = [] # Se almacenan las octavas.
duraciones = [] # Se almacenan las duraciones.

# El usuario inserta el nombre del archivo
print("Inserte el nombre del archivo de texto: ")
nombreTxt = input()

# Lectura del archivo
with open(nombreTxt) as archivo:
    for linea in archivo: # Se lee cada linea
        lineaAct = linea # Se define una nueva variable para actualizar la linea.
        # Se elminan los espacios o tabluaciones que están al inicio de la línea.
        patron = "^( |\t)+"
        lineaAct = re.sub(patron, '', lineaAct)
        # Se elminan los espacios o tabluaciones que están al final de la línea.
        patron = "( |\t)+$"
        lineaAct = re.sub(patron, '', lineaAct)
        # Si no hay comentarios al inicio de la linea se empieza a buscar por los demás lexemas.
```

```
        if len(re.findall(r"^#.+", lineaAct)) == 0:
            cantEspacio = linea.count(' ') + linea.count('\t') # Se cuentan los espacios y tabluaciones para saber cuantas veces habrá que analizar la linea.
            for i in range(cantEspacio + 1):
                # Al analizar la linea se van buscando por lexemas y elminando los espacios para que el analizador de léxico (scanner) no tenga problemas.
                token = Tokenizador(lineaAct) # Se llama al tokenizador.
                patron = "^\S+( |\t)+"
                lineaAct = re.sub(patron, '', lineaAct)
                #print(lineaAct)
        # En caso contrario solamente se busca por el comentario sin eliminar espacios ni tabulaciones que se encuentren en el resto de la linea.
        else:
            token = Tokenizador(lineaAct) # Se llama al tokenizador.



# Se imprime la lista de tokens y lexemas.
print(*tokens)
print(*lexemas)

print("\nLos lexemas son correctos.")

Parser(tokens) # Se llama al analizador de sintaxis.
```

```
# Llamados para traducir o converitr las notas de MUT a números para que los pueda leer la librería de sounddevice.
traduccionNota(lexemas)
traduccionOctava(lexemas)
traduccionDuracion(lexemas)

# Imprime las listas.
print(*notas)
print(*octavas)
print(*duraciones)

# Reproducelas notas ya parametrizadas para que la librería de sounddevice las reconozca.
for n, o, d in zip(notas, octavas, duraciones):
    play(n, o, d)
```

Entradas

- **Entradas Correctas**
  - bass A0q A1q C#1q C#2q | D1q D2q B0q E1q |
    bass A0q A1q C#2q
    #BAR
    bass D1q D2q B0q E1q
    #BAR
  - #TITLE J. S. Bach ltrlell-Tempered Clavier, I. , Prelude no. 11
    #VOICES left right
    # first measure
    right F4s C4s A3s G3s A3s C4s F3s A3s C4s Eb4s D4s C4s
    left F2e A2e C3e A2e F2e A2e
    right D4s Bb3s F3s E3s F3s Bb3s D3s F3s A3s C4s Bb3s A3s
    left Bb2e D3e Bb2e F1q Re
    #BAR
- **Entradas para detectar errores**
  - **Error de Lexema**
    - bass A0 A1q C#1q C#2q | D1q D2q B0q E1q |
      bass A0q A1q C#2q
      #BAR
      bass D1q D2q B0q E1q
      #BAR
  - **Error de Sintaxis**
    - bass A0q A1q C#1q C#2q | D1q D2q B0q E1q | bass
      bass A0q A1q C#2q
      #BAR
      bass D1q D2q B0q E1q
      #BAR

Link del Video:

https://drive.google.com/file/d/1w33eDIJbgEbCW0lYkRfPeXURTMQP8R-w/view?usp=sharing

Experiencia de Aprendizaje y Resultados Obtenidos

Fue una experiencia retadora pero muy satisfactoria de completar porque pude comprender y utilizar lo aprendido en clase. Primeramente tuve que realizar las expresiones regulares para poder identificar los lexemas de manera correcta, lo cual en este caso pudo ser aplicable ya que la gramática de las notas es regular, como lo describía la Figura 13.

Las ventajas de las expresiones regulares es que son fáciles de implementar, sin embargo, la desventaja es que éstas se utilizan solamente para lenguajes regulares.

Para ello tuve que tener el conocimiento previo de la Jerarquía de Lenguajes de Chomsky, donde en esta Situación Problema se abordaron los lenguajes regulares (el cual es MUT), así como los que son libre de contexto, lo cual sirvió para tratar la sintaxis de MUT, ya que al diseñar la gramática BNF, ésta debe ser libre de contexto para poder aplicar el método de Descenso Recursivo, cuya ventaja es que es fácil de implementar, pero la desventaja es que éste no es muy rápido a comparación de otros métodos.

También debo decir que me gustó haber trabajado con música y elementos como las frecuencias, notas, así como el proceso de convertir el lenguaje de MUT para que lo leyera la librería de sound device para que fuera capaz de reproducir sonido.

Los resultados que obtuve fueron satisfactorios porque el programa fue capaz de identificar los lexemas, la sintaxis y convertir las notas en sonidos, así como también identificar los errores de lexemas y sintaxis.

Quizá si se hubiera trabajado con otros métodos el programa hubiera sido más eficiente, convirtiéndolo más rápido, aunque puede que hubiese sido más complicado de codificar.

Referencias Bibliográficas

John Ortiz Ordoñez. (18 de Mayo de 2019). “Python 3 - Receta 40: Tokenizar una Cadena de Texto usando Expresiones Regulares”. [Video]. Recuperado de https://youtu.be/_Mm2QZFA0cM

John Ortiz Ordoñez. (09 de Octubre de 2019). “Python - Ejercicio 84: Contar la Cantidad de Ocurrencias de un Carácter en una Cadena de Caracteres”. [Video]. Recuperado de https://youtu.be/KWPtPU0uKDk

John Ortiz Ordoñez. (08 de Marzo de 2020). “Python Curso V2: 67: insert() para Agregar un Elemento en una Lista en una Posición Específica”. [Video]. Recuperado de https://youtu.be/KJpIffzPAQI

John Ortiz Ordoñez. (25 de Junio de 2020). “Python - Ejercicio 878: Remover Caracteres en Mayúscula desde una Cadena con una Expresión Regular”. [Video]. Recuperado de https://youtu.be/_Pxeo3GvnEM