Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación.

IC5701 – Compiladores e Intérpretes.

Proyecto 2 Explorador – Ciruelas Extendido

Profesor: Aurelio Sanabria.

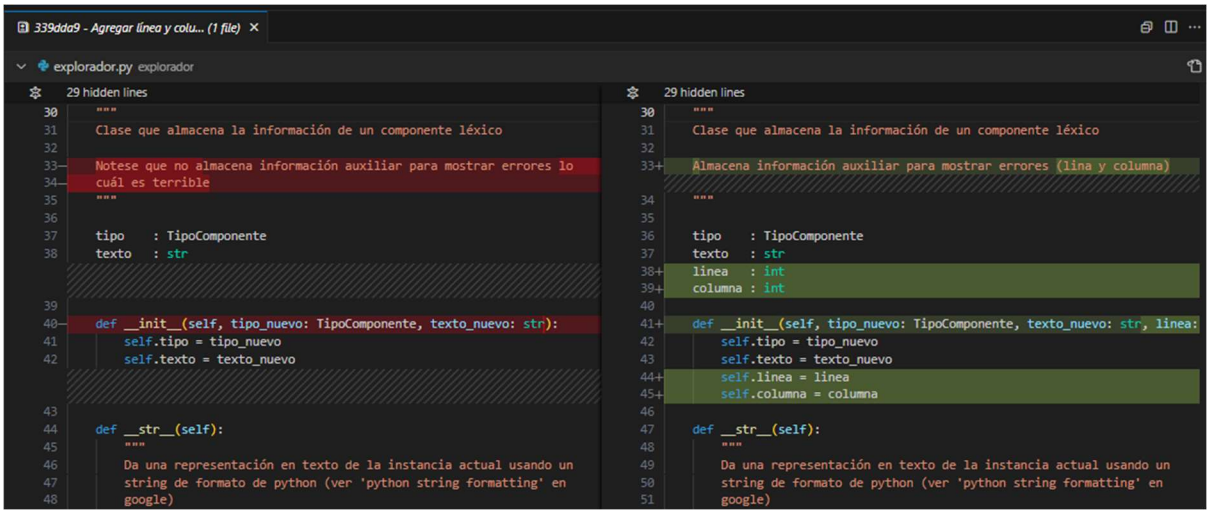
Estudiante:

Tamara Nicole Rodríguez Luna.

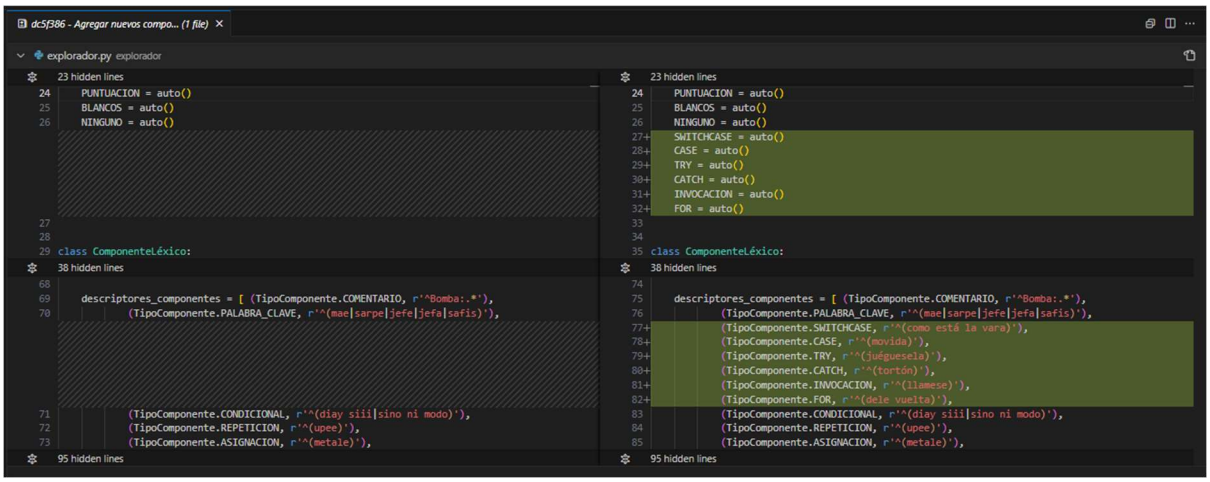
19 de abril del 2025.

Demostración y discusión

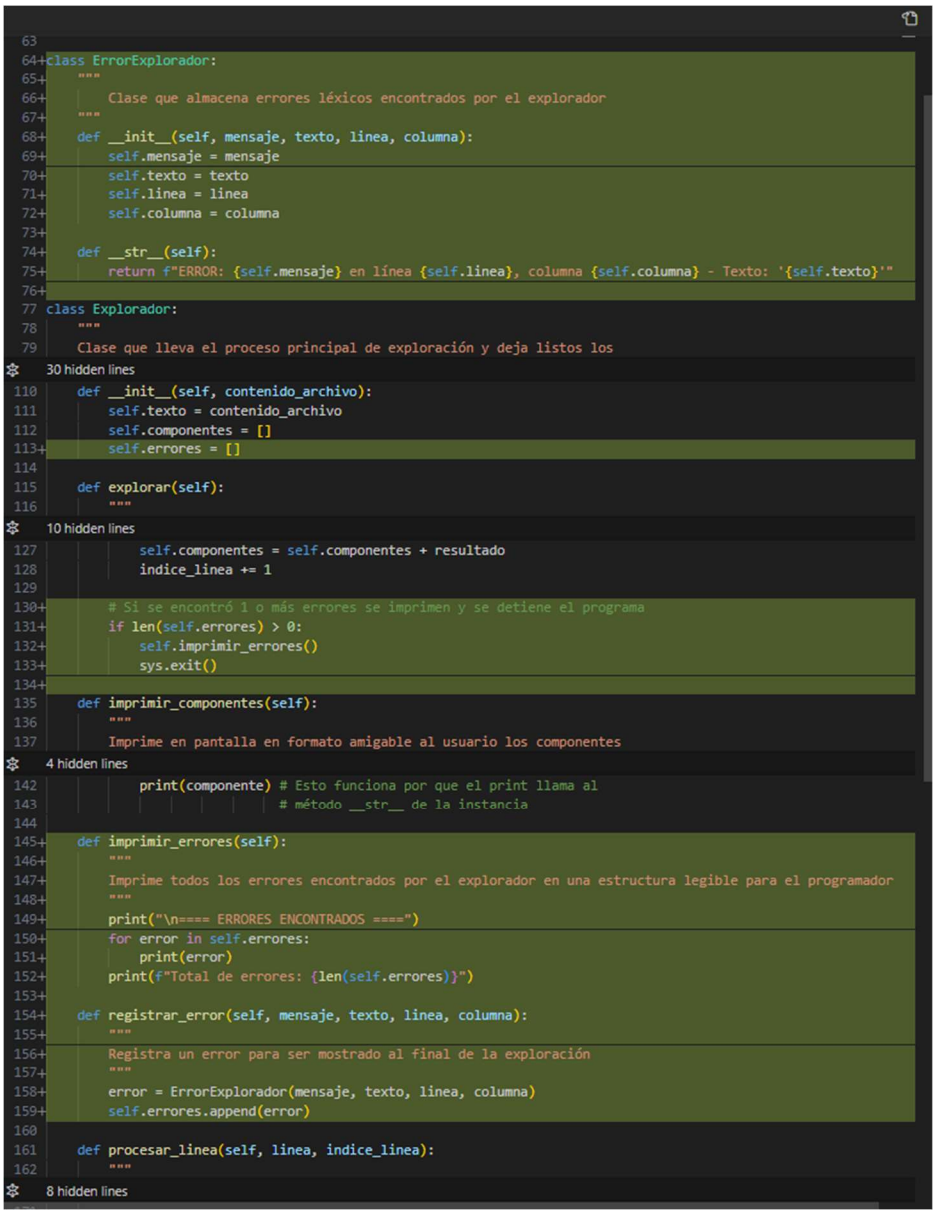
En Ciruelas el explorador no almacenaba la línea y columna de los componentes, por lo tanto, los agregué para poder mostrar errores más precisos, de paso adaptando el comentario para que esté acorde con el código



Seguidamente añadí los nuevos componentes léxicos de la gramática que definí para ciruelas

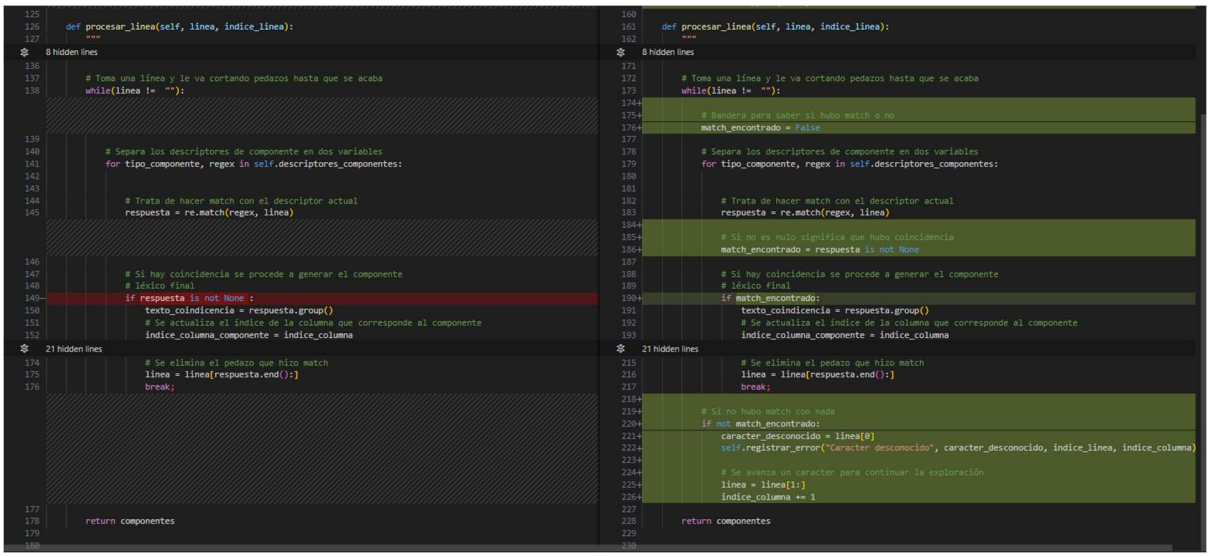


Luego creé una clase ErrorExplorador para poder almacenar los posibles errores que puede dar el explorador, así puedo irlos almacenando en una lista e imprimir todos los errores cuando el explorador termine

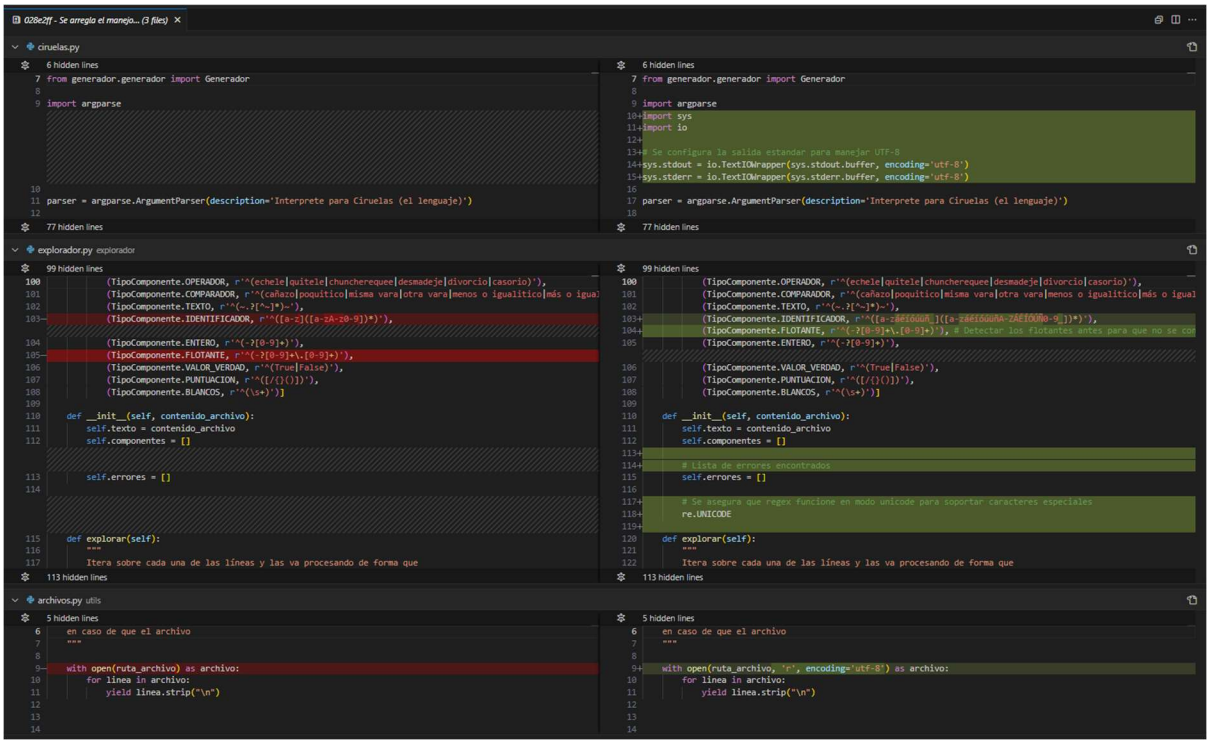


```
63
64+class ErrorExplorador:
65+    """
66+        Clase que almacena errores léxicos encontrados por el explorador
67+    """
68+    def __init__(self, mensaje, texto, linea, columna):
69+        self.mensaje = mensaje
70+        self.texto = texto
71+        self.linea = linea
72+        self.columna = columna
73+
74+    def __str__(self):
75+        return f"ERROR: {self.mensaje} en línea {self.linea}, columna {self.columna} - Texto: '{self.texto}'"
76+
77 class Explorador:
78     """
79     Clase que lleva el proceso principal de exploración y deja listos los
   30 hidden lines
110     def __init__(self, contenido_archivo):
111         self.texto = contenido_archivo
112         self.componentes = []
113+        self.errores = []
114
115     def explorar(self):
116         """
   10 hidden lines
127             self.componentes = self.componentes + resultado
128             indice_linea += 1
129
130+        # Si se encontró 1 o más errores se imprimen y se detiene el programa
131+        if len(self.errores) > 0:
132+            self.imprimir_errores()
133+            sys.exit()
134+
135     def imprimir_componentes(self):
136         """
137         Imprime en pantalla en formato amigable al usuario los componentes
   4 hidden lines
142             print(componente) # Esto funciona por que el print llama al
143                               # método __str__ de la instancia
144
145+    def imprimir_errores(self):
146+        """
147+        Imprime todos los errores encontrados por el explorador en una estructura legible para el programador
148+        """
149+        print("\n==== ERRORES ENCONTRADOS ====")
150+        for error in self.errores:
151+            print(error)
152+        print(f"Total de errores: {len(self.errores)}")
153+
154+    def registrar_error(self, mensaje, texto, linea, columna):
155+        """
156+        Registra un error para ser mostrado al final de la exploración
157+        """
158+        error = ErrorExplorador(mensaje, texto, linea, columna)
159+        self.errores.append(error)
160
161     def procesar_linea(self, linea, indice_linea):
162         """
   8 hidden lines
```

El único error que pude identificar es cuando el caracter no corresponde a ninguno de los ya establecidos



Por alguna razón, Ciruelas original me estaba dando error al tratar de procesar los caracteres con tildes o diéresis o la ñ, para solucionar eso hice que re funcione en unicode, agregué la codificación utf-8 y todos los caracteres especiales del español a las expresiones regulares, es algo que ya había contemplado en mi documento de gramática. Por otro lado, en este mismo commit, vi que el orden de las expresiones regulares importa, tuve que mover la expresión regular de los flotantes antes de los enteros, sino todos los números los detectaba como enteros.



Ciruelas es un lenguaje que me parece muy similar a Python en su ausencia de ; o algún otro símbolo para terminar una línea, aún así Ciruelas sigue utilizando las llaves para delimitar una función, por ejemplo, lo que en Python veríamos así:

```
def imprimir(texto):
```

```
print(texto)
```

En Ciruelas se vería así:

```
mae imprimir (texto){

sueltele(texto)

}
```

Esto entiendo que es por la simplicidad de buscar un símbolo específico para saber que ya terminó una función, hacer eso es mucho más seguro que el manejo de indentación que hace Python. Pero al centrarnos en las funciones nuevas que estoy proponiendo en Python tenemos el try-except que se ve así:

```
try:

resultado = a / b

except:

print("No puedes dividir por cero")
```

Mientras que en Ciruelas mi propuesta es que se vea así, es menos limpio con respecto a cantidad de caracteres, pero más fácil de procesar:

```
juéguesela{

resultado metale a desmadeje b

}tortón{

sueltele(~No puedes dividir por cero~)

}
```

Además en Python hay un for que lo podemos ver en el siguiente ejemplo:

```
for i in range(len(texto)):

print(texto[i])
```

El de Ciruelas que propongo pienso que es bastante diferente, pero queda más claro todas las variables involucradas y el "step" del for, que en este caso sube de 1 en 1:

```
dele vuelta (i metale 0 / i poquitico largo_texto / i metale i
hechele 1){

sueltele(viene_bolita(texto / i))

}
```

Por último, el switch-case es extraño de comparar puesto que nada similar existía antes de Python 3.10, es curioso que se tardara tanto en implementar, sí técnicamente no hace falta, pero muchas veces se termina viendo más limpio que otras opciones como solo implementaciones alternativas de tratar de hacer lo mismo únicamente con if, diccionarios o funciones, luego se implementó el match que tiene la siguiente forma:

```
match x:

case 1:

print("El resultado es 1")

case 2:

print("El resultado es 2")

case _:

print("El resultado es diferente a 1 y 2")
```

En la extensión de Ciruelas yo propuse que tuviera una forma similar, pues no me agradan los breaks que se pueden encontrar en los switches de otros lenguajes, pero com más llaves para saber dónde empieza y dónde termina cada caso, como se muestra a continuación:

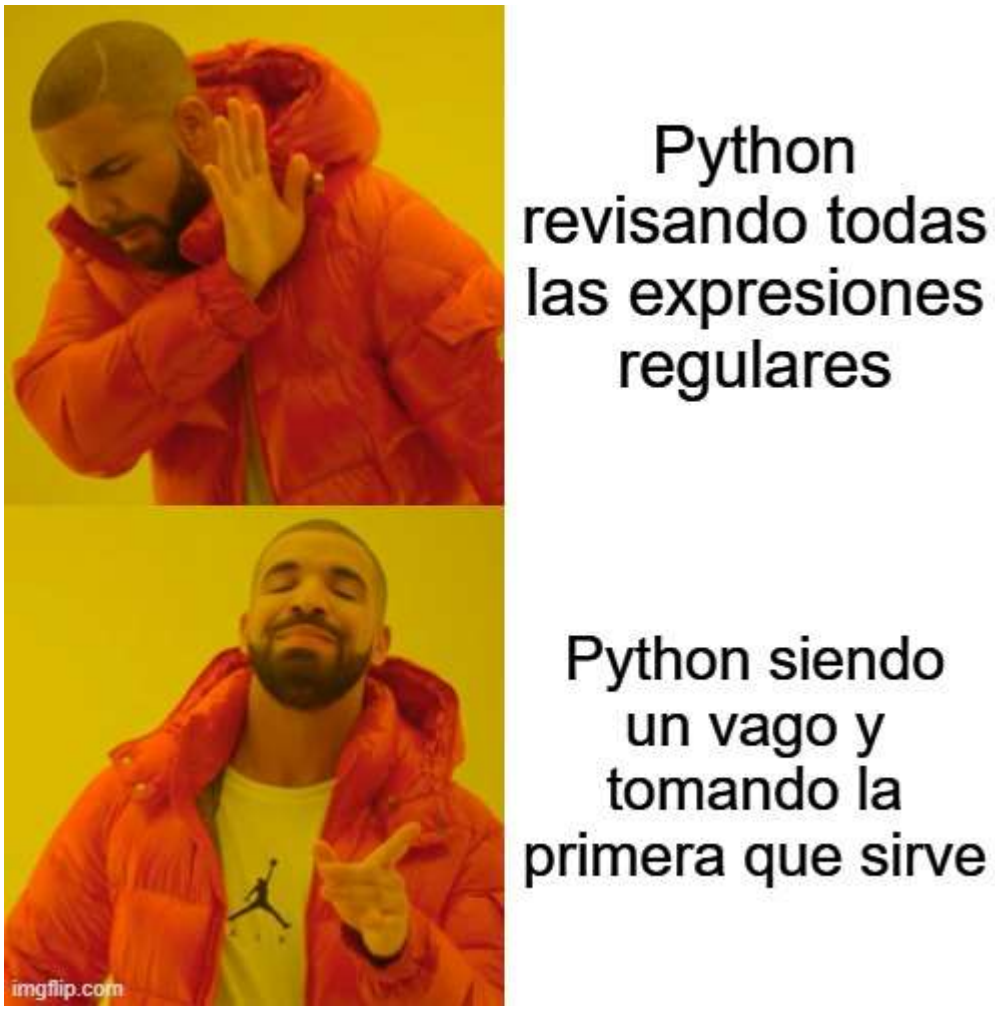
```
como está la vara(x){

movida 1{

sueltele(~El resultado es 1~)

     }

movida 2{

sueltele(~El resultado es 2~)

     }

sino ni modo{

sueltele(~El resultado es diferente a 1 y 2~)

     }

}
```

Lecciones aprendidas

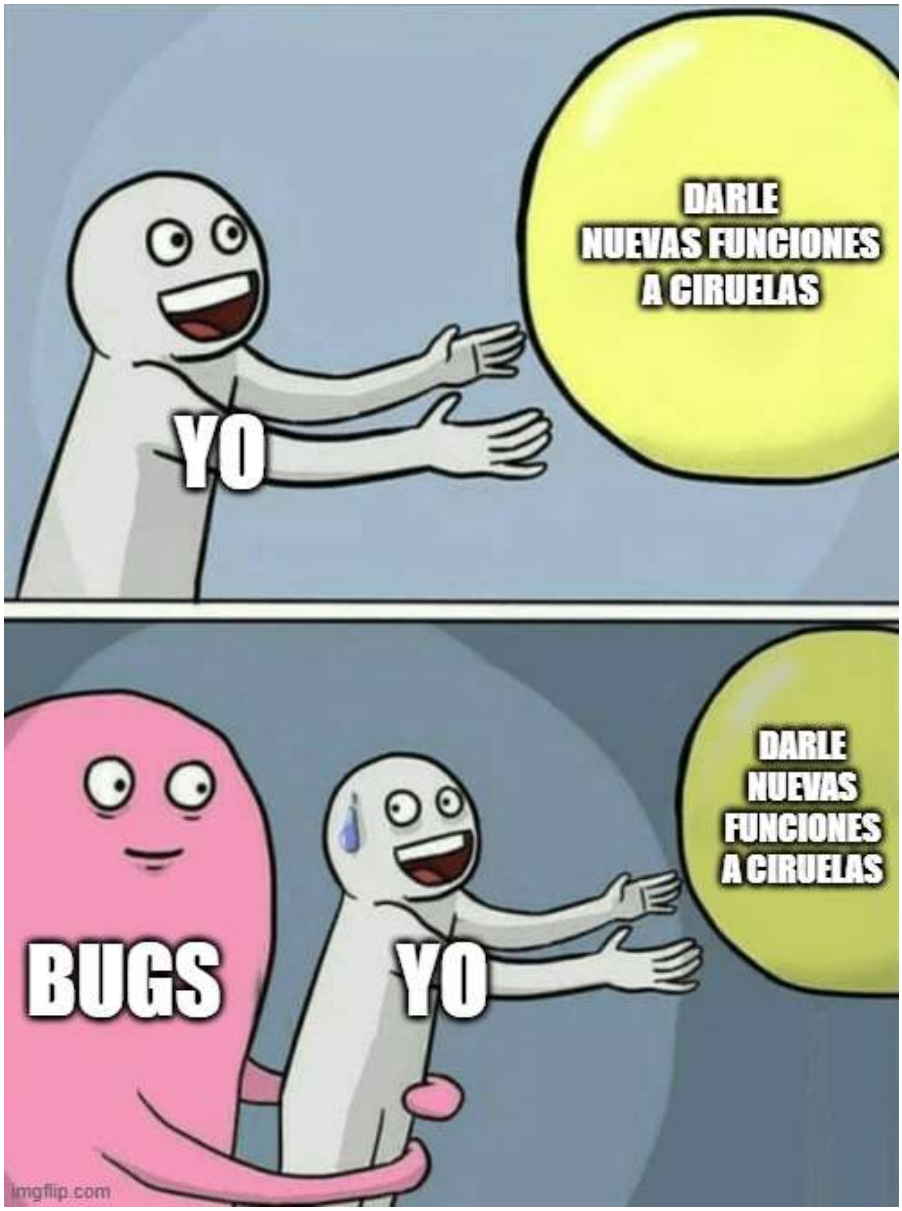
Lo que voy aprendiendo de esto es primeramente no fiarse del código ya hecho, no hay que optimizar absolutamente todo si no hace falta para evitar un problema mayor, pero el caso de los caracteres especiales del español cuando vi que no se procesaron supe que tenía que

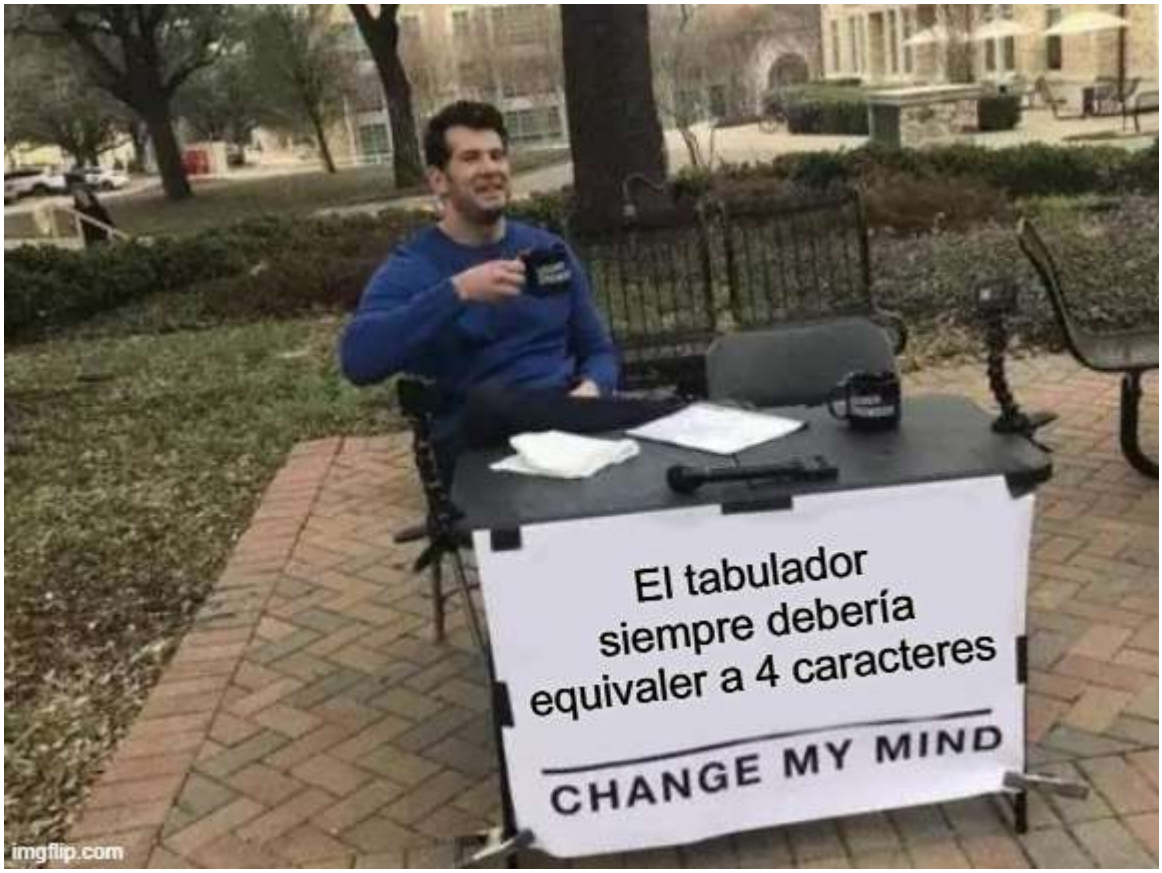
modificar lo que ya estaba hecho, igualmente cuando vi que el explorador asumía que todos los números eran enteros únicamente porque esa definición estaba antes que la de flotante, entiendo que eso pasa porque el intérprete de Python funciona de arriba hacia abajo y como vio un match no se molestó en verificar los demás. Por último, me sorprendió que el manejo de columnas fuera tan raro, pensé que era un estandar, pero cuando vi las inconsistencias entre la columna que me indicaba Visual Studio Code y el block de notas pude aprender que cada editor de texto considera al tabulador de formas diferentes, algunos lo ven como solo 1 caracter, otros como 2, 4 u 8 caracteres, esa era la razón, entonces tuve que decidir un número que fue 4, principalmente para seguir lo que indica VS Code.

Memes




Modificar el código de Ciruelas
Solo concentrarme en lo nuevo
YO
JAKE-CLARK.TUMBLR
imgflip.com


DARLE NUEVAS FUNCIONES A CIRUELAS
YO
DARLE NUEVAS FUNCIONES A CIRUELAS
BUGS
YO
imgflip.com


El tabulador
siempre debería
equivaler a 4 caracteres
CHANGE MY MIND
imgflip.com


Mejorar
la
gramática
de Ciruelas
Hacer
mi repo
basado
en Ciruelas
Empezar
a
arreglar bugs
Empezar
a
arreglar bugs
imgflip.com