



Objetivos

General

- Desarrollar un analizador léxico eficiente para un lenguaje de programación inspirado en Python, capaz de reconocer y clasificar correctamente los diferentes componentes léxicos presentes en el código fuente

Específico

- Implementar el análisis léxico para un lenguaje de programación **inspirado** en Python
- Implementar lógica y reglas de clasificación para cada tipo de token definido
- Graficar cada token presente en el código fuente

Descripción

El programa Mitnick lo ha contratado para realizar la primera fase del proyecto inspirado en Python, el cual se denomina parser-py. La fase consiste en crear un escáner en el cual se deben de reconocer y clasificar componentes léxicos de un programa, como identificadores, operadores, constantes y palabras clave.

Identificadores

Los identificadores en Python son nombres utilizados para identificar variables, funciones, clases u otros objetos definidos por el usuario. Algunas reglas para los identificadores en Python son:

- Deben comenzar con una letra (a-z o A-Z) o un guión bajo (_).
- No pueden comenzar con un número.
- Pueden contener letras, números y guiones bajos.
- parser-py es sensible a mayúsculas y minúsculas, por lo que los identificadores "miVariable" y "MIVARIABLE" serían diferentes.

Ejemplo de identificadores:



- mi_variable
- nombre
- edad
- _total

Operadores en Python:

Los operadores en Python son símbolos especiales utilizados para realizar operaciones matemáticas, lógicas y de manipulación de datos. Algunos de los operadores más comunes en Python son:

Aritméticos		
Nombre	Símbolo	Observación
Suma	+	
Resta	-	
Exponente	**	
División	/, //	
Módulo	%	
Multiplicación	*	
Comparación		
Nombre	Símbolo	Observación
Igual	==	
Diferente	!=	
Mayor que	>	
Menor que	<	



Mayor o igual que	>=	
Menor o igual que	<=	
Lógicos		
Nombre	Símbolo	Observación
y	and	
o	or	
negación	not	
Asignación		
Nombre	Símbolo	Observación
Asignación	=	También es posible hacer cualquier combinación con un operador aritmético ejemplo <ul style="list-style-type: none">• *= , multiplica y asigna• -=
Palabras clave		
Nombre	Símbolo	Observación
Palabra reservada	<ul style="list-style-type: none">• and• as• assert• break• class• continue• def• del• elif• else• except• False• finally• for	Es importante tener en cuenta que no se deben usar palabras clave como nombres de variables o funciones, ya que Python las reconoce como parte de la sintaxis y generaría un error en el código.



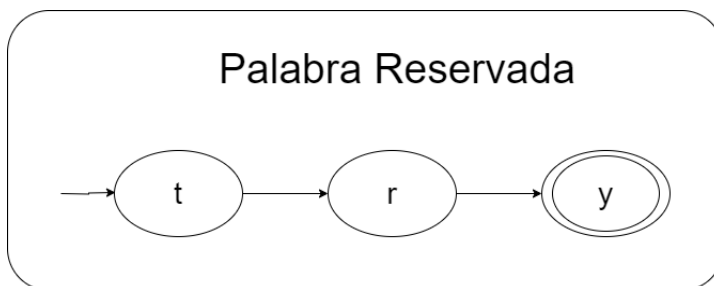
	<ul style="list-style-type: none"> • from • global • if • import • in • is • lambda • None • nonlocal • not • or • pass • raise • return • True • try • while • with • yield 	
Constantes		
Nombre	Símbolo	Observación
Entero	1,2...n	
Decimal	1.4, 0.001	
Cadena	"una cadena", 'cadena 123'	una cadena de caracteres dentro de comillas simples o dobles
booleanas	True, False	
Comentario		
Nombre	Símbolo	Observación
Comentario	#es tes un comentario	solo se tomará en cuenta los comentarios de una línea



Otros		
Nombre	Símbolo	Observación
Paréntesis	(,)	
Llaves	{ , }	
Corchetes	[,]	
Coma	,	
Punto y coma	;	
Dos puntos	:	

Características adicionales

- Cada token deberá guardar la línea y columna de su ubicación
- Se deberá de graficar cada lexema el cual deberá estar identificado a que token pertenece, mostrando la línea y columna donde se encuentra ejemplo:



- No se deberá de verificar si el código es correcto
- Se deberá mostrar si existen errores, los errores son tokens que no coinciden encajan en alguna categoría
- Cada token debe colorearse según la siguiente tabla



Token	Color
Identificadores	blanco/negro
Operadores Aritméticos	Celeste
Comparación	
Lógicos	
Asignación	
Palabras clave	Morado
Constantes	Rojo/Anaranjado
Comentarios	Gris
Otros	verde

Reporte

Se deberá de listar los tokens identificados, el reporte debe incluir **Token**, **Patron**, **Lexema**, **línea**, y **columna**, ejemplo:

Token	Patrón	Lexema	Línea	Columna
Palabra_Reservada	try	try	1	10
Palabra_Reservada	try	try	1	12
ID	([\\w] _)+(\\w \\d)*	_fs	2	1
Otro	((1	2



Interfaz de usuario

Los siguientes mockups presentan un ejemplo de como se puede realizar la interfaz, pero el diseño queda a criterio de cada uno

```
1 # Este es un programa sencillo para calcular el área de un triángulo
2
3 # Función para calcular el área del triángulo
4 def calcular_area(base, altura):
5     area = (base * altura) / 2
6     return area
7
8 try:
9     base = float(input("Ingrese el valor de la base del triángulo: "))
10    altura = float(input("Ingrese el valor de la altura del triángulo: "))
11
12    # Verificar que los valores ingresados sean válidos (mayores a 0)
13    if base > 0 and altura > 0:
14        area_triángulo = calcular_area(base, altura)
15        print("El área del triángulo es:", area_triángulo)
16    else:
17        print("Error: Los valores deben ser mayores a 0.")
```

Error

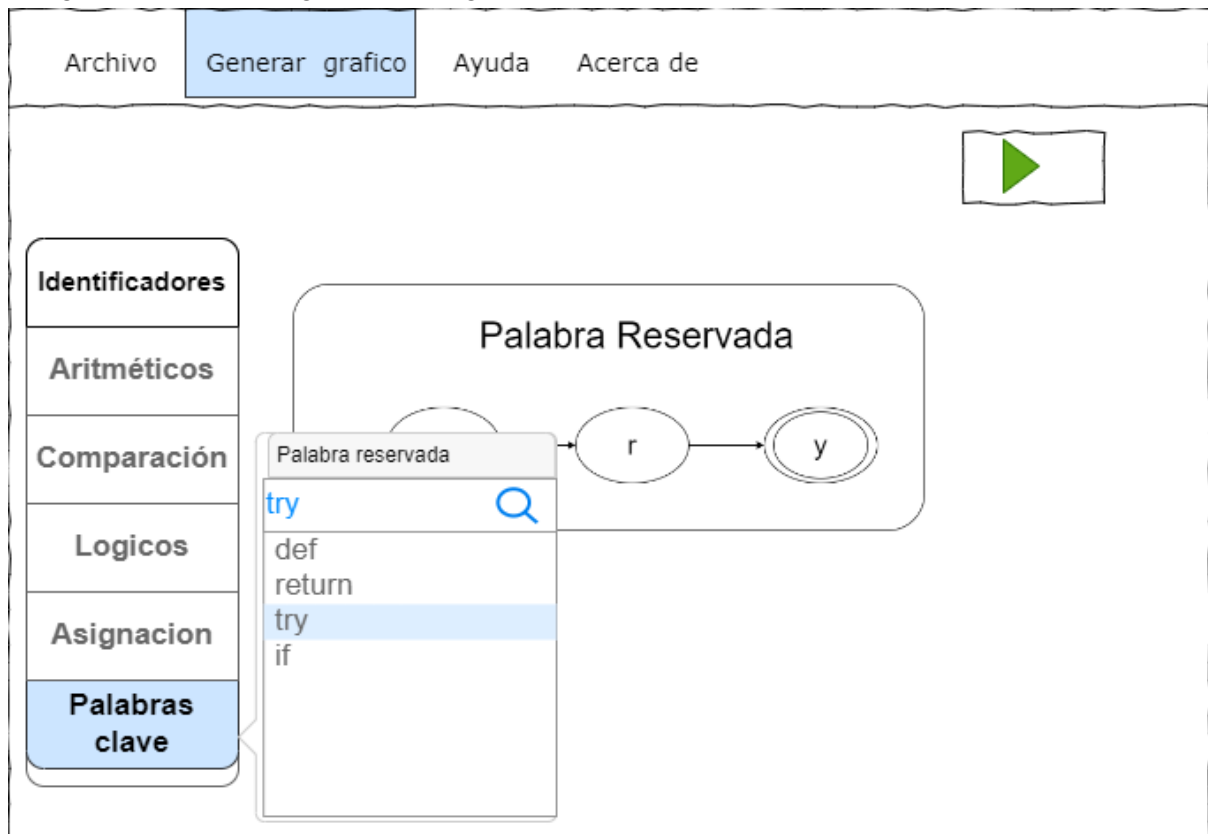
Editor

La interfaz de usuario debe incluir un editor de texto que muestre la línea y la columna donde se encuentra el puntero. Este editor debe permitir cargar un archivo de texto o escribir directamente en su área de trabajo



Generar Gráfico

Las gráficas se deben generar con graphviz



Importante


- Debe dividirse en frontend y backend
- Lenguaje recomendado Java (backend)
- Es permitido usar cualquier librería/framework/lenguaje para el frontend
- se recomienda usar graphviz para genera las imágenes
- Requisito para la práctica 2
- Usar gitflow
- Es válido usar cualquier IDE o editor de texto
- Copias obtendrán nota de cero, se notificará al docente y a coordinación, no tienen derecho a la siguiente práctica
- **No es permitido usar expresiones regulares en el lenguaje de programación**



Entrega

- Enlace de repositorio
- En la rama principal deberá tener una carpeta con el ejecutable
- Manuales
- Fecha: 22/08/2023 20:59:00

Calificación

- Hoja preliminar de calificación
-  Hoja de calificación

Archivo de Entrada

- [Ejemplo del archivo de entrada](#)