

## Examen Parcial

### Patrick Andres Ramirez Santos

DOCENTE	CARRERA	CURSO
MSc. Vicente Enrique Machaca Arceda	Escuela Profesional de Ingeniería de Software	Compiladores

## Índice

<b>1. Introduccion a Swift Script v. 1989.0</b>	<b>2</b>
1.1. Justificacion . . . . .	2
1.2. Objetivos . . . . .	2
<b>2. Propuesta</b>	<b>2</b>
2.1. Especificación léxica . . . . .	2
2.1.1. Definición de los comentarios . . . . .	2
2.1.2. Definición de los identificadores . . . . .	2
2.1.3. Definición de las palabras clave . . . . .	2
2.1.4. Definición de los literales . . . . .	3
2.1.5. Definición de los operadores . . . . .	3
2.2. Expresiones regulares . . . . .	4
2.3. Ejemplos de código . . . . .	4
2.3.1. Hola mundo . . . . .	4
2.3.2. Factorial iterativo . . . . .	4
2.3.3. Factorial recursivo . . . . .	5
<b>3. Gramatica</b>	<b>5</b>
<b>4. Implementacion</b>	<b>6</b>

# 1. Introduccion a Swift Script v. 1989.0

## 1.1. Justificacion

Este documento presenta la creación de un lenguaje de programación inspirado en la discografía de Taylor Swift. La música de Taylor Swift es conocida por su narrativa emocional y lírica, lo que proporciona una base rica para la construcción de un lenguaje de programación. Este lenguaje busca proporcionar una forma novedosa y atractiva de aprender los conceptos de programación y compilación, al tiempo que se explora la intersección entre la música y la ingeniería de software.

## 1.2. Objetivos

El objetivo principal de este proyecto es diseñar e implementar un lenguaje de programación basado en la discografía de Taylor Swift. Los objetivos específicos son los siguientes:

- Desarrollar una especificación léxica y sintáctica para el lenguaje.
- Implementar un compilador que pueda traducir programas escritos en este lenguaje a un lenguaje de programación de alto nivel.
- Proporcionar ejemplos de código y documentación detallada para ayudar a los usuarios a aprender y utilizar este lenguaje.

# 2. Propuesta

## 2.1. Especificación léxica

### 2.1.1. Definición de los comentarios

Listing 1: Comentarios

```
shake Este seria un comentario de una linea
shakeitoff
    Este seria
    Un comentario
    multilinea
shakeitoff
```

### 2.1.2. Definición de los identificadores

El language utiliza la palabra enchanted para definir una variable y los tipos se expresan al final como en Typescript

Listing 2: Identificadores

```
enchanted variable_bool : meetyou
enchanted variable_string : wonderstruck
enchanted variable_double = 15 : thpage
```

### 2.1.3. Definición de las palabras clave

e dentro de un loop representa el iterador y ee el break para for

Listing 3: For loop

```
me 1 e 12 {  
    speaknow(e)  
    ee  
}
```

oohoooh sirve como break, al igual que en el for es combinar el inicio del bucle 2 veces

Listing 4: While loop

```
ooh variable > 10 {  
    speaknow("Hiii")  
    oohoooh  
}
```

Se utilizan las eras para preguntar por condicionales, si estas en tu Lover era es el inicio de if, sino puede ser Red era y el ultimo recurso es Reputation era.

Listing 5: If-Else-Elif

```
loverera condition_if {  
    speaknow("hiii")  
} redera elif_condition {  
    speaknow("hiii")  
} repera {  
    speaknow("Else condition")  
}
```

Se utiliza la referencia a Speak Now para imprimir en consola y toma como argumento un tipo wonderstruck (string)

Listing 6: Imprimir en consola

```
enchanted variable_string = "Hola Mundo" : wonderstruck  
speaknow("hiii")  
speaknow(variable_string)
```

Listing 7: Definir funcion

```
isme nombre_funcion (att1: tipo1, att2: tipo2 ...) : tipo{  
    hi valor_retorno  
} imtheproblem
```

#### 2.1.4. Definición de los literales

Solo se permitira el uso de comillas dobles para cualquier tipo de string (wonderstruck), pero ademas se incluyen los valores sparksfly y badblood para definir verdadero y falso respectivamente

#### 2.1.5. Definición de los operadores

Se utilizaran los operadores +, -, \*, / y el de porcentaje. Estos simbolos tendran la misma funcionalidad que cualquier otro lenguaje de programacion, ademas los parentesis mantienen su funcionalidad junto a los simbolos = y ==.

## 2.2. Expresiones regulares

Token	Expresion regular
identificador	[a z][a Z0 9]
literal	([a-Z]+ [0-9]+)+
numeral	[0 9]+    [0 9]+ . [0 9]+
oper plus	numeral + numeral
oper mul	numeral * numeral
oper sus	numeral * numeral
oper division	numeral * numeral
men	>
may	<
plus	>
minus	<
times	>
divide	<
assign	=
compare	==
compare <sub>or</sub>	
compare <sub>and</sub>	
Comentarios	shake
Comentario bloque	shakeitoff expression shakeitoff
function	isme (expression)* imtheproblem
return	hi
end function	imtheproblem
if	loverera (expression ) { (expression)* }
else	repera { (expression)* }
elif	redera (expression ) { (expression)* }
while	ooh ( expression ) {expression* }
break while	oohoooh
for	me (numeral) e (numeral) {expression* }
break for	ee
boolean	meetyou (type _bool)
type _bool	(SparksFly   BadBlood)
double	thpage identificador
string	wonderstruck "(identificador) "
rigth_p	(
left_p	)

Tabla 1: Tabla de tokens y expresiones regulares

## 2.3. Ejemplos de código

### 2.3.1. Hola mundo

```
speaknow("Hola Mundo")
```

### 2.3.2. Factorial iterativo

```
isme factorial1 (number: thpage) : thpage{
    enchanted total = 0 : thpage
    me 1 e number{
        total *= e
    }
    hi total
} imtheproblem
```

### 2.3.3. Factorial recursivo

```
isme factorial2 (number: thpage) : thpage{
    loverera number == 0 {
        hi 1
    } repera {
        hi number * factorial2(number - 1)
    }
} imtheproblem
```

## 3. Gramatica

```
P -> AS
P -> CS
P -> IS

IS -> while( CE ) { E }
IS -> for( CE ) { E }

CS -> loverera( CE ) { E } CSELIF
CSELIF -> nulo
CSELIF -> CELSE
CSELIF -> redera( CE ) { E } CELSE
CSELSE -> repera { E }

CE -> E CEX
CEX -> == E
CEX -> != E
CEX -> < E
CEX -> > E

AS -> enchanted identificador X
X -> = E : TY
X -> : TY

TY -> thpage
TY -> wonderstruck
TY -> twenty
TY -> meetYou

E -> P
```

```

E -> T E'
E' -> + T E'
E' -> - T E'
E' -> && T E'
E' -> || T E'

E' -> nulo
T -> F T'
T' -> * F T'
T' -> / F T'
T' -> nulo

F -> FX FF
FF -> nulo
FF -> newLINE P

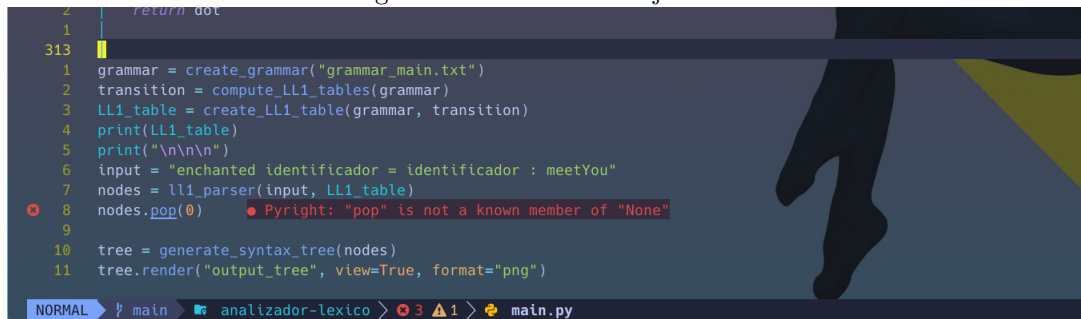
FX -> ( E )
FX -> identificador
FX -> numeral
FX -> type_bool
FX -> comilla string comilla

type_bool -> SparksFly
type_bool -> BadBlood

```

## 4. Implementacion

Figura 1: Parametros de ejecucion



```

1  return dot
2
313
1  grammar = create_grammar("grammar_main.txt")
2  transition = compute_LL1_tables(grammar)
3  LL1_table = create_LL1_table(grammar, transition)
4  print(LL1_table)
5  print("\n\n\n")
6  input = "enchanted identificador : meetYou"
7  nodes = ll1_parser(input, LL1_table)
8  nodes.pop(0)  • Pyright: "pop" is not a known member of "None"
9
10 tree = generate_syntax_tree(nodes)
11 tree.render("output_tree", view=True, format="png")

```

NORMAL ▶ main ▶ analizador-lexico > 3 1 > main.py

Figura 2: Tabla de transición

```
python main.py
E': {'==', '!=', '!', '}', '>', '<'}
T: {'==', '!=', '!', '}', '>', '<', '&&', '-', '||', '<'}
S: set()
P: {'==', '!=', '!', '}', '>', '<', '&&', '-', '||', '<'}
T': {'==', '!=', '!', '}', '>', '<', '&&', '-', '||', '<'}
FX: {'==', '!=', '!', '}', '>', '<', '*', 'newLINE', '&&', '/', '!', '||', '<'}
AS: {'==', '!=', '!', '}', '>', '<', '*', '&&', '/', '!', '||', '<'}

Finalizado Follow!
X      : TY
CS      : loverera( CE ) { E } CSELIF
type_bool
CE      E CEX      E CEX      BadBlood
TY      E CEX
CSELIF  nulo      nulo      nulo      nulo      CSELIF  nulo      nulo
CEX      != E
IS      nulo      nulo      FX FF      FX FF
FF      nulo      nulo      nulo      nulo
F      nulo      nulo      nulo      nulo
CEELSE  nulo      nulo      nulo      nulo
E      P      T E'      T E'
E'      nulo      nulo      F T'      F T'
T      P
S      P $
P      AS
T'      nulo      nulo      numeral type_bool
FX      enchanted identificador X
AS      nulo      nulo      nulo      nulo

[19 rows x 34 columns]
```

Figura 3: Arbol y Parser

