Práctica 06

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

PRÁCTICA	TEMA	DURACIÓN
06	Expresiones Regulares en Python	3 horas

1. Datos de los estudiantes

- Grupo: 1
- Git Hub: `https://github.com/CrazyDani17/practica6_compiladores`
- Cuaderno de códigos: `https://github.com/CrazyDani17/practica6_compiladores/blob/master/practica6_compiladores.ipynb`
- Integrantes:
  - Guillermo Alemán
  - Marvik Del Carpio
  - Daniel Mendiguri
  - Daniela Vilchez

2. Competencias del curso

- Conocer las bases de la teoría de la computación para la implementación de un compilador.
- Implementar un compilador para un lenguaje de baja complejidad.

3. Competencias de la práctica

- Entender como convertir un autómata finito no determinista a un autómata finito determinista.

4. Equipos y materiales

- Latex
- Conección a internet
- Python
- Ply

5. Entregables

- Se debe elaborar un informe en **Latex** donde se responda a cada ejercicio de la Sección 7.
- Por cada 5 minutos de retraso, el alumno tendrá un punto menos.

6. Marco teórico

Python ofrece una librería para reconocer expresiones regulares (re). Este sigue el estandar para reconocer expresiones regulares, en la Tabla 1 mostramos algunas reglas, esto fue extraido de la documentación.

Tabla 1: Expresiones regulares en Python.

Pattern	Description
^	Matches beginning of line
$	Matches end of line
.	Any single character except newline.
$a*$	Matches 0 or more occurrences of preceding expression
$a+$	Matches 1 or more occurrence of preceding expression
$a?$	Matches 0 or 1 occurrence of preceding expression
$a\{n\}$	Matches exactly n number of occurrences of preceding expression
$a\{n,\}$	Matches n or more occurrences of preceding expression
$a\{n,m\}$	Matches at least n and at most m occurrences of preceding expression
$a\|b$	Matches either a or b
[^abc]	Any character except a or b or c
[^0-9]	Any non-digit character

Así es como se escribe un código en Python para saber, si una cadena es aceptada por una expresión regular:

```
import re

pattern = '^a...s$'
test_string = 'abysrs'
result = re.match(pattern, test_string)

if result:
   print("Search successful.")
else:
   print("Search unsuccessful.")
```

Así es como se escribe un código en Python, para que retorne todas las coincidencias de una expresión regular:

```
import re

string = """Hello my Number is 123456789 and
my friend's number is 987654321"""
# A sample regular expression to find digits.
regex = r'\d+'

match = re.findall(regex, string)
print(match)
```

7. Ejercicios

1. Escriba el código Python de **(6 puntos)**:

   - Un programa que reconozca los números flotantes.

```
import re

pattern = '(0|[1-9][0-9]*)\.[0-9]*'
test_string = input("Into a float number: ")
result = re.match(pattern, test_string)

if result:
 print("Search successful.")
else:
 print("Search unsuccessful.")
```

   - Un programa que reconozca la fechas en este formato dd-mm-yyyy.

```
import re

pattern = '([0-2][0-9]|30|31)\-(0[0-9]|11|12)\-[0-9]{4}'
test_string = input("Into a date: ")
result = re.match(pattern, test_string)

if result:
 print("Search successful.")
else:
 print("Search unsuccessful.")
```

   - Un programa que reconozca los espacios en blanco. incluyendo las tabulaciones y saltos de linea.

```
import re

pattern = '(\t|\n|\s)+'
test_string = input("Into spaces, endlines and tabs: ")
result = re.match(pattern, test_string)

if result:
 print("Search successful.")
else:
 print("Search unsuccessful.")
```

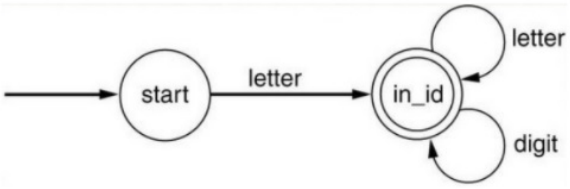
   - Un programa que reconozca los identificadores del lenguaje propuesto para el trabajo final.

```
import re

pattern = '[a-zA-Z]+[\w]*'
test_string = input("Into a identifier in Llama: ")
result = re.match(pattern, test_string)

if result:
 print("Search successful.")
else:
 print("Search unsuccessful.")
```

2. Normalmente para reconocer una expresión regular se siguen los siguientes pasos:

   - Convertir la expresión regular a un Automata Finito No Determinista (AFND).
   - El AFND debe ser transformado a un Automata Finito Determinista (AFD).
   - Finalmente este AFD, es representado mediante una tabla de transiciones.
   - Se implementa un programa para reconocer las ocurrencias de una expresión regular utilizando la tabla de transiciones.

   En esta caso, le brindamos el AFD para reconocer identificadores, se le pide obtener la tabla de transiciones e implementar un programa en Python que tome esta tabla (la puedes definir estáticamente) y reconozca los identificadores de un archivo de texto (similar a la función *findall* de python). **(11 puntos)**



Solución:

```
import re

def caracter(chacha):
   global Fin
   Fin=""
   letra='[a-zA-Z]'
   digito='\d'

   #comparamos si es digito o letra
   if(re.match(letra,chacha)):
       return 'letter'
   else:
       if(re.match(digito,chacha)):
           return 'digit'
       #si no es ni un digito ni un operador entonces es un caracter no validp
       return -1

def valida(chacha):
 A={'start':{'letter':'in_id','digit':'empty'},'in_id':{'letter':'in_id','digit':'in_id'}}
 state='start'
 for i in chacha:
   if caracter(i)==-1:
     return
   state=A[state][caracter(i)]
   if state=='empty':
     break
 if state=='in_id':
   return chacha

file=open('archivo.txt','r')
texto=file.readlines()
file.close()

def encuentra_todo(texto):
   ocurrencias=[]
   for renglon in texto:
     for palabra in renglon.split(' '):
       aux = valida(palabra)
       if aux != None:
         ocurrencias.append(aux)
   return ocurrencias

print(encuentra_todo(texto))
```

8. Rúbricas

Rúbrica	**Cumple**	**Cumple con obs.**	**No cumple**
Informe: El informe debe estar en Latex, con un formato limpio (buena presentación) y facil de leer.	3	1.5	0
Implementación: Responde cada ejercicio correctamente.	17	7.5	0
Errores ortográficos: Por cada error ortográfico, se le descontará un punto.	-	-	-