Práctica: Expresiones regulares en Python y Bash

Adriana Pérez Espinosa

6 de febrero de 2026

1. Introducción

El módulo `re` en Python contiene funciones y clases relativas a búsqueda de patrones mediante expresiones regulares. En las búsquedas se puede hacer uso de caracteres predefinidos y metacaracteres:

Caracteres predefinidos

- `\d` Cualquier carácter que sea dígito
- `\D` Cualquier carácter que no sea dígito
- `\w` Cualquier carácter letra o dígito
- `\W` Es el complemento de `\w`
- `\s` Espacio en blanco
- `\S` Cualquier carácter que no sea espacio

Metacaracteres

- `+` El carácter de la izquierda aparecerá una o varias veces
- `*` El carácter de la izquierda aparecerá cero o más veces
- `?` El carácter de la izquierda aparecerá cero o una vez
- `{}` Indica el número de veces que debe aparecer el carácter de la izquierda

2. Funciones del módulo `re`

Python ofrece la descripción de este módulo así como el uso de los metacaracteres mediante: `help(re)`.

Dos funciones principales son `search` y `findall`. La función `re.search()` busca un patrón desde el principio de la cadena y devuelve un objeto Match con información sobre la búsqueda.

Listing 1: Ejemplo de la función search

```
import re
"""LA FUNCION SEARCH BUSCA LA PRIMER OCURRENCIA EN UNA CADENA"""
cadena = "pos = 100 + vel*500;"
x = re.search("ve", cadena)
print("La cadena analizada: " + x.string)
print("Indice donde inicia la cadena buscada:", x.start())
print("Indice donde termina la cadena buscada:", x.end())
print("Indice donde inicia y termina la cadena buscada:", x.span())
print("Devuelve la cadena encontrada:", x.group())
```

Salida:

```
La cadena analizada: pos = 100 + vel*500;
Indice donde inicia la cadena buscada: 12
Indice donde termina la cadena buscada: 14
Indice donde inicia y termina la cadena buscada: (12, 14)
Devuelve la cadena encontrada: ve
```

¿Qué hace `re.search("ve", cadena)`?

La función `re.search("ve", cadena)` busca la **primera aparición** de la subcadena `"ve"` dentro de la cadena analizada.

En la cadena:

```
"pos = 100 + vel*500;"
```

la subcadena `"ve"` aparece dentro del fragmento `"vel*500"`.

Conteo de índices (muy importante)

En Python, los índices de las cadenas **empiezan en 0**, no en 1. Esto significa que el primer carácter de una cadena se encuentra en la posición 0.

A continuación se muestra la cadena carácter por carácter con su índice correspondiente:

Índice	Carácter
0	p
1	o
2	s
3	espacio
4	=
5	espacio
6	1
7	0
8	0
9	espacio
10	+
11	espacio
12	v **(inicio de "ve")**
13	e
14	l
15	*
16	5
17	0
18	0
19	;

Por esta razón, el método `x.start()` devuelve el valor **12**, ya que indica la posición donde inicia la coincidencia encontrada.

Otros ejemplos

A continuación se muestran algunos patrones comunes en expresiones regulares y el significado del resultado que devuelven al aplicarse sobre una cadena.

Uso del patrón `\d`

El patrón `\d` busca cualquier dígito numérico. La función `re.search` devuelve la primera coincidencia encontrada en la cadena.

```
x = re.search(r"\d", cadena)
print("Devuelve la cadena encontrada:", x.group())
```

La salida es:

```
Devuelve la cadena encontrada: 1
```

Esto ocurre porque el carácter `1` es el primer dígito que aparece en la cadena analizada.

Uso del patrón \w

El patrón `\w` representa cualquier carácter alfanumérico, incluyendo letras, dígitos y el carácter guion bajo.

```
x = re.search(r"\w", cadena)
print("Devuelve la cadena encontrada:", x.group())
```

La salida es:

```
Devuelve la cadena encontrada: p
```

Esto se debe a que el carácter `p` es el primer carácter alfanumérico que aparece en la cadena, y por lo tanto es la primera coincidencia del patrón `\w`.

Función `findall` en Python

La función `findall` del módulo `re` se utiliza para **encontrar todas las coincidencias** de un patrón dentro de una cadena.

A diferencia de `re.search`, que devuelve únicamente la primera coincidencia, `findall` devuelve **todas las apariciones** que cumplan con el patrón especificado.

Listing 2: Ejemplo de la función findall

```
import re
cadena = "pos = 100 + vel*500;"
print("Cadena:", cadena)
x = re.findall("\d", cadena)
print(x)
```

Salida:

```
Cadena: pos = 100 + vel*500;
['1', '0', '0', '5', '0', '0']
```

Patrón: `\d+`

El patrón `\d+` indica **uno o más dígitos consecutivos**. A diferencia del patrón `\d`, que identifica un solo dígito, `\d+` agrupa secuencias completas de números.

Al aplicar este patrón sobre la cadena analizada, se identifican los números completos presentes en ella.

```
['100', '500']
```

Esto ocurre porque en la cadena aparecen dos secuencias de dígitos consecutivos: `100` y `500`, las cuales son devueltas como elementos independientes dentro de la lista.

3. Otros elementos importantes del módulo `re`

El módulo `re` incluye diversos elementos que permiten construir expresiones regulares más flexibles y expresivas. A continuación se presentan algunos de los más utilizados:

- **Corchetes** `[]`: permiten definir **conjuntos o rangos** de caracteres (por ejemplo, `[a-z]` o `[0-9]`).
- **Paréntesis** `()`: se utilizan para **agrupar caracteres** y controlar el alcance de los operadores.
- **Punto** `.`: funciona como un **comodín**, ya que coincide con cualquier carácter (excepto el salto de línea).
- **Barra invertida** `\textbackslash`: se emplea para **escapar caracteres especiales** y darles un significado literal.
- **Barra vertical** `|`: representa una **alternativa**, indicando que puede coincidir uno u otro patrón.

4. Ejemplo de análisis léxico sobre un archivo Java

En este ejemplo se utiliza un archivo escrito en Java como entrada para aplicar distintas expresiones regulares en Python, con el objetivo de **identificar patrones léxicos comunes** como identificadores, palabras reservadas y operadores.

Archivo de entrada: `Factorial.java`

Listing 3: Archivo Factorial.java

```
class Factorial {
    public static void main(String args[]) {
        int i, factn = 1;
        int numero = 5;
        for (i = 1; i <= numero; i++) {
            factn = factn * i;
        }
        System.out.println("El factorial de" + numero + "es:" + factn);
    }
}
```

Uso de expresiones regulares en Python

El siguiente programa en Python lee el archivo `Factorial.java` y aplica diferentes patrones de expresiones regulares para mostrar cómo se reconocen distintos tipos de elementos que contiene el programa en Java.

Listing 4: Expresiones regulares aplicadas a un archivo Java

```
import re

file = open('Factorial.java', mode='r')
archivo = file.read()

print("Patr n:-[a-z]")
x = re.findall("[a-z]", archivo)
print(x)

print("Patr n:-[a-z][a-z]")
x = re.findall("[a-z][a-z]", archivo)
print(x)

print("Patr n:-[a-z]+")
x = re.findall("[a-z]+", archivo)
print(x)

print("Patr n:-[A-Z][a-z]+")
x = re.findall("[A-Z][a-z]+", archivo)
print(x)

print("Patr n:-[A-Za-z][A-Za-z0-9_]*")
x = re.findall("[A-Za-z][A-Za-z0-9_]*", archivo)
print(x)

print("Patr n:-<=|>=|<|>")
x = re.findall("<=|>=|<|>", archivo)
print(x)

file.close()
```

Actividad

Observe la salida generada por cada patrón y realice un **reporte breve** explicando qué reconoce cada expresión regular y por qué se obtiene dicha salida. Incluya **capturas de pantalla de la ejecución** como evidencia.

5. Ejercicio

Considere el siguiente programa escrito en lenguaje C, el cual realiza el cálculo del área de un círculo a partir del valor del radio ingresado por el usuario:

Listing 5: Cálculo del área de un círculo

```c
#include<stdio.h>
#include<conio.h>
main()
{
    float radio, area;
    area = 0;
    radio = 0;
    printf("Area de un circulo");
    // entrada de datos
    printf("\nIntroduce el radio =>");
    scanf("%f", &radio);
    /* proceso para calcular el Area total */
    area = 3.1416 * pow(radio, 2);
    // salida
    printf("\n\nArea = %5.2f", area);
    getch();
}
```

5.1. Actividad: Reconocimiento de elementos de un programa en C

A partir del código anterior, desarrolle un programa en Python que **identifique y clasifique** los siguientes elementos léxicos presentes en el archivo fuente:

- **Identificadores** (por ejemplo, `var`, `total1`, `_x`).
- **Números enteros** (por ejemplo, `0`, `15`, `1024`).
- **Números reales** (por ejemplo, `3.14`, `0.5`, `10.0`).
- **Cadenas** delimitadas por comillas dobles (por ejemplo, `"hola"`).
- **Comentarios**:
  - de una línea (`// ...`)
  - multilínea (`/* ... */`)
- **Operadores aritméticos** (`+ - * / % =`).
- **Operadores relacionales** (`> < >= <=`).
- **Operadores lógicos** (`&&` y `||`).

El programa deberá mostrar la salida de forma clara, indicando el tipo de elemento léxico reconocido y el elemento o elementos reconocidos correspondiente.

6. Expresiones regulares en BASH

En el entorno BASH, las expresiones regulares se utilizan para **buscar, filtrar, validar y procesar texto**. Estas expresiones se aplican comúnmente sobre archivos de texto, por lo que resulta fundamental contar con un archivo de entrada para observar su funcionamiento.

Archivo de texto de ejemplo

Considere el siguiente contenido, el cual deberá guardarse en un archivo llamado `datos.txt`:

```
usuario1
admin
user_23
123
variable
total = 45
if (x <= 10)
```

Este archivo servirá como entrada para los ejemplos de expresiones regulares en BASH que se muestran a continuación.

Herramientas que utilizan expresiones regulares

En BASH, las expresiones regulares se emplean principalmente a través de las siguientes herramientas:

- `grep`: búsqueda de patrones en archivos.
- `sed`: búsqueda y reemplazo de texto.
- `awk`: procesamiento de texto basado en patrones.
- `[[ ]]`: validación de cadenas dentro de scripts.

Todas estas herramientas procesan el texto **línea por línea**.

Uso de expresiones regulares con `grep`

El comando `grep` permite buscar patrones dentro de un archivo.

Ejemplo: líneas que contienen solo letras

```
grep -E "^[a-zA-Z]+$" datos.txt
```

Este comando muestra únicamente las líneas formadas exclusivamente por letras, como `admin` y `variable`.

Ejemplo: líneas que contienen números

```
grep -E "[0-9]+" datos.txt
```

En este caso, se muestran las líneas que contienen uno o más dígitos, como `usuario1`, `123` y `total = 45`.

Uso de expresiones regulares en estructuras condicionales

BASH permite utilizar expresiones regulares directamente en condiciones para validar cadenas.

```
cadena="user_23"

if [[ $cadena =~ ^[a-zA-Z_][a-zA-Z0-9_]*$ ]]; then
    echo "Identificador válido"
else
    echo "Identificador inválido"
fi
```

Este ejemplo verifica si una cadena cumple con la forma de un identificador válido.

Uso de `sed`

El comando `sed` permite realizar sustituciones sobre el archivo de entrada utilizando expresiones regulares.

```
sed 's/[0-9]\+/NUM/g' datos.txt
```

Este comando reemplaza todas las secuencias de números por la cadena `NUM`.

Uso de `awk`

`awk` combina expresiones regulares con acciones.

```
awk '/<=/' datos.txt
```

Este comando imprime las líneas que contienen el operador relacional `<=`.

7. Ejercicio: Uso de expresiones regulares en BASH

Objetivo

Aplicar expresiones regulares en BASH para analizar información del sistema, en particular el uso de espacio en disco, a partir de la salida de comandos del sistema operativo.

Descripción

El comando `df -h` muestra información sobre el uso de disco en el sistema. Esta información se presenta en forma de texto, por lo que puede ser procesada mediante expresiones regulares.

En este ejercicio se desarrollará un **script en BASH** que utilice expresiones regulares para identificar particiones con alto uso de disco.

Actividad

Desarrolle un script en BASH que realice las siguientes tareas:

- Muestre el uso de disco del sistema utilizando el comando `df -h`.
- Utilice expresiones regulares para identificar los porcentajes de uso de disco en la salida del comando.
- Filtre y muestre únicamente las particiones cuyo uso de disco sea mayor o igual al 80 %.

Indicaciones

- El script deberá ejecutarse desde la terminal.
- Elabore un **reporte breve** explicando:
  - qué información del sistema se está analizando,
  - qué reconoce cada expresión regular,
  - y por qué se obtiene la salida mostrada.
- Incluya **capturas de pantalla** de la ejecución como evidencia.

Nota

Este ejercicio muestra que las expresiones regulares no solo se utilizan para el reconocimiento de lexemas, sino también para el análisis y procesamiento de información generada por el sistema operativo.

8. Entregables

Fecha límite de entrega: 14 de febrero de 202 a las 11:59 pm

- Una carpeta comprimida ZIP que contiene:
- Archivos relacionados con el ejercicio en python
- Archivos relaciones con el ejercicio en bash
- Reporte con las capturas y argumentaciones correspondientes.