

Guía Completa de la Librería RE (Expresiones Regulares) de Python

¿Qué son las Expresiones Regulares?

Las expresiones regulares (regex) son patrones que se utilizan para buscar, validar y manipular texto de manera eficiente. La librería `re` de Python proporciona todas las herramientas necesarias para trabajar con estos patrones.

```
python

import re # Importar la librería
```

Funciones Principales de la Librería RE

1. `re.match()`

Busca una coincidencia AL INICIO de la cadena.

```
python

resultado = re.match(patron, cadena, flags=0)
```

Ejemplo:

```
python

import re

patron = r'[A-Z][a-z]+'
texto = "Hola mundo"

resultado = re.match(patron, texto)
if resultado:
    print(f"Coincidencia encontrada: {resultado.group()}") # Output: Hola
else:
    print("No hay coincidencia")
```

Características:

- Solo busca al principio de la cadena
- Retorna un objeto Match si encuentra coincidencia, None si no

- Útil para validar formatos específicos

2. re.search()

Busca la PRIMERA coincidencia en cualquier parte de la cadena.

```
python

resultado = re.search(patron, cadena, flags=0)
```

Ejemplo:

```
python

import re

patron = r'\d+' # Busca números
texto = "Tengo 25 años y peso 70 kg"

resultado = re.search(patron, texto)
if resultado:
    print(f"Primer número encontrado: {resultado.group()}") # Output: 25
```

Diferencia con match():

- `match()` solo busca al inicio
- `search()` busca en toda la cadena

3. re.findall()

Encuentra TODAS las coincidencias y las retorna como una lista.

```
python

lista_coincidencias = re.findall(patron, cadena, flags=0)
```

Ejemplo:

```
python
```

```
import re

patron = r'\d+'
texto = "Tengo 25 años, peso 70 kg y mido 180 cm"

numeros = re.findall(patron, texto)
print(numeros) # Output: ['25', '70', '180']
```

Características:

- Retorna una lista con todas las coincidencias
- Si no encuentra nada, retorna lista vacía []
- Muy útil para extraer múltiples elementos

4. re.finditer()

Similar a findall(), pero retorna un iterador de objetos Match.

```
python

iterador = re.finditer(patron, cadena, flags=0)
```

Ejemplo:

```
python

import re

patron = r'\w+'
texto = "Hola mundo Python"

for match in re.finditer(patron, texto):
    print(f"Palabra: {match.group()}, Posición: {match.start()}-{match.end()}")
```

Ventajas:

- Proporciona más información (posición, grupos, etc.)
- Eficiente en memoria para textos grandes

5. re.sub()

Reemplaza las coincidencias con un texto nuevo.

```
python
```

```
nuevo_texto = re.sub(patron, reemplazo, cadena, count=0, flags=0)
```

Ejemplo:

```
python
```

```
import re
```

```
patron = r'\d+'
```

```
texto = "Tengo 25 años"
```

```
nuevo_texto = re.sub(patron, 'XX', texto)
```

```
print(nuevo_texto) # Output: "Tengo XX años"
```

Parámetros:

- `count`: número máximo de reemplazos (0 = todos)
- Puede usar funciones como reemplazo

6. re.subn()

Como sub(), pero retorna una tupla (nuevo_texto, número_de_reemplazos).

```
python
```

```
resultado, cantidad = re.subn(patron, reemplazo, cadena, count=0, flags=0)
```

Ejemplo:

```
python
```

```
import re
```

```
patron = r'\d+'
```

```
texto = "25 años y 70 kg"
```

```
resultado, cantidad = re.subn(patron, 'XX', texto)
```

```
print(f"Texto: {resultado}, Reemplazos: {cantidad}") # Output: Texto: XX años y XX kg, Reemplazos: 2
```

7. re.split()

Divide una cadena usando un patrón como separador.

```
python
```

```
lista = re.split(patron, cadena, maxsplit=0, flags=0)
```

Ejemplo:

```
python

import re

patron = r'[;,]' # Separar por comas o punto y coma
texto = "manzana,pera,uva,plátano"
frutas = re.split(patron, texto)
print(frutas) # Output: ['manzana', 'pera', 'uva', 'plátano']
```

8. re.compile()

Compila un patrón para reutilizarlo múltiples veces (más eficiente).

```
python

patron_compilado = re.compile(patron, flags=0)
```

Ejemplo:

```
python

import re

patron = re.compile(r'\d+')
texto1 = "Tengo 25 años"
texto2 = "Peso 70 kg"

print(patron.search(texto1).group()) # Output: 25
print(patron.search(texto2).group()) # Output: 70
```

Ventajas:

- Mejor rendimiento cuando se usa el mismo patrón múltiples veces
- El objeto compilado tiene los mismos métodos (match, search, etc.)

Metacaracteres y Patrones Comunes

Metacaracteres Básicos:

- `.` : Cualquier carácter (excepto nueva línea)
- `^` : Inicio de cadena
- `$` : Final de cadena
- `*` : 0 o más repeticiones
- `+` : 1 o más repeticiones
- `?` : 0 o 1 repetición
- `{n}` : Exactamente n repeticiones
- `{n,m}` : Entre n y m repeticiones

Clases de Caracteres:

- `[abc]` : a, b, o c
- `[a-z]` : Cualquier letra minúscula
- `[A-Z]` : Cualquier letra mayúscula
- `[0-9]` : Cualquier dígito
- `[^abc]` : Cualquier cosa EXCEPTO a, b, o c

Secuencias Especiales:

- `\d` : Dígito (equivale a `[0-9]`)
- `\D` : No dígito
- `\s` : Espacio en blanco
- `\S` : No espacio en blanco
- `\w` : Carácter de palabra (letra, dígito o `_`)
- `\W` : No carácter de palabra

NOTA: En el contexto de la tarea, está PROHIBIDO usar `\w` y sus variantes.

Flags (Modificadores)

Los flags modifican cómo se comporta la expresión regular:

```
python
```

```
import re
```

Flags comunes:

re.IGNORECASE # o re.I - Ignora mayúsculas/minúsculas

re.MULTILINE # o re.M - ^ y \$ funcionan con cada línea

re.DOTALL # o re.S - . incluye nueva línea

re.VERBOSE # o re.X - Permite comentarios en el patrón

Ejemplo:

```
python
```

```
import re
```

```
patron = r'python'
```

```
texto = "Me gusta PYTHON"
```

Sin flag

```
resultado1 = re.search(patron, texto)
```

```
print(resultado1) # Output: None
```

Con flag IGNORECASE

```
resultado2 = re.search(patron, texto, re.IGNORECASE)
```

```
print(resultado2.group()) # Output: PYTHON
```

Grupos y Captura

Grupos de Captura:

```
python
```

```
import re
```

```
patron = r'(\d{4})-(\d{2})-(\d{2})'
```

```
texto = "Fecha: 2024-08-06"
```

```
match = re.search(patron, texto)
```

```
if match:
```

```
    print(f"Fecha completa: {match.group(0)}") # 2024-08-06
```

```
    print(f"Año: {match.group(1)}")           # 2024
```

```
    print(f"Mes: {match.group(2)}")           # 08
```

```
    print(f"Día: {match.group(3)}")           # 06
```

Grupos No Capturadores:

```
python
```

```
patron = r'(?:(Sr|Sra)\. ([A-Z][a-z]+)' # (?...) no captura
```

Objeto Match

Cuando una función encuentra una coincidencia, retorna un objeto Match con métodos útiles:

```
python
```

```
import re
```

```
patron = r'(\w+)@(\w+\.\w+)'
```

```
texto = "Correo: usuario@ejemplo.com"
```

```
match = re.search(patron, texto)
```

```
if match:
```

```
    print(f"Coincidencia completa: {match.group()}") # usuario@ejemplo.com
```

```
    print(f"Grupo 0: {match.group(0)}") # usuario@ejemplo.com
```

```
    print(f"Grupo 1: {match.group(1)}") # usuario
```

```
    print(f"Grupo 2: {match.group(2)}") # ejemplo.com
```

```
    print(f"Posición inicio: {match.start()}") # Posición donde inicia
```

```
    print(f"Posición final: {match.end()}") # Posición donde termina
```

```
    print(f"Span: {match.span()}") # Tupla (inicio, final)
```

Ejemplos Prácticos para la Tarea

1. Validación de Caracteres Permitidos:

```
python
```



```

import re

def validar_verso(verso):
    # Patrón basado en EBNF del documento
    patron = r'^[A-Za-zÑñÁÉÍÓÚáéíóú0-9¿?¡!.,:()\-"]+$'
    return bool(re.match(patron, verso))

# Ejemplo de uso:
verso1 = "Hola mundo, ¿cómo estás?"
verso2 = "Hola mundo @$%" # Contiene caracteres no permitidos

print(validar_verso(verso1)) # True
print(validar_verso(verso2)) # False

```

2. Extracción de Vocales:

```

python

import re

def extraer_vocales(palabra):
    vocales = re.findall(r'[aeiouáéíóú]', palabra.lower())
    return "".join(vocales)

palabra = "hermoso"
print(extraer_vocales(palabra)) # "eoo"

```

3. Limpieza de Signos de Puntuación:

```

python

import re

def limpiar_palabra(verso):
    # Remover signos de puntuación al final
    palabra_limpia = re.sub(r'[¿?¡!.,:()\-"]+$', "", verso.strip())
    return palabra_limpia

verso = "¡Hola mundo!"
print(limpiar_palabra(verso)) # "¡Hola mundo"

```

Consejos y Buenas Prácticas

1. Usar Raw Strings:

```
python

# Incorrecto (puede causar problemas)
patron = '\d+'

# Correcto
patron = r'\d+'
```

2. Compilar Patrones Reutilizables:

```
python

import re

# Si vas a usar el mismo patrón múltiples veces
patron_email = re.compile(r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}')

# Luego puedes usarlo múltiples veces
emails = ["test@example.com", "user@domain.org"]
for email in emails:
    if patron_email.match(email):
        print(f'{email} es válido')
```

3. Manejar Casos Especiales:

```
python

import re

def buscar_patron_seguro(patron, texto):
    try:
        resultado = re.search(patron, texto)
        return resultado.group() if resultado else None
    except re.error:
        print("Error en el patrón de expresión regular")
        return None
```

Aplicación en el Proyecto Juez de Freestyle

En el contexto de la tarea, las expresiones regulares se utilizan para:

1. **Validar caracteres permitidos** en los versos

2. **Extraer palabras finales** eliminando signos de puntuación
3. **Identificar vocales** para rimas asonantes
4. **Analizar sufijos** para rimas consonantes
5. **Procesar el archivo de entrada** separando palabras clave y estrofas

La librería `re` es fundamental para cumplir con el requerimiento de usar exclusivamente expresiones regulares para la detección de patrones en el texto.