¿Qué son las *regex* o *expresiones regulares*?

Las regex son una serie de patrones y reglas que los gobiernan que sirven básicamente para hacer búsquedas en textos o evaluar si determinada *string* o cadena de caracteres cumple las condiciones que nos interesen. Muy útiles para trabajar de manera avanzada con texto. Algunas de sus utilidades:

- Buscar y reemplazar en código
- Validar texto de entrada
- Renombrar archivos
- Buscar archivos en la línea de comandos
- Buscar en bases de datos
- Hacer scraping
- Limpiar texto

Algunas cosas a tener en cuenta sobre las regex:

- Es una tecnología vieja, inventada en los 50.
- Son multidisciplinares. ¿Quién no ha necesitado alguna vez usar regex, da igual que sea de front, de back, de investigación...
- Están muy extendidas, hay muchos programas que los soportan. Por supuesto, todos los lenguajes de programación, pero también programas como Word, Writer, AntConc, TshwaneLex... Lo "malo" es que, por esto mismo, también hay muchos "dialectos". Aquí vamos a aprenderlas en general, lo que es común a todos los dialectos, y en especial en Python.

Caracteres literales y metacaracteres

Caracteres literales

Corresponden con caracteres de la string tal cual. Si ponemos `y` nos encuentra la primera `y` de `Monty Python`, aunque depende de las opciones del programa o la función que estemos usando al programar.

Uso de Expresiones Regulares en Python

En este ejemplo, utilizamos el módulo `re` de Python para realizar búsquedas en una cadena de texto utilizando expresiones regulares.

```
import re

# Busca la primera ocurrencia de la letra 'y' en la cadena "Monty
Python"
a = re.search("y", "Monty Python")

# Encuentra todas las ocurrencias de la letra 'y' en la cadena "Monty
Python"
```

```python
b = re.findall("y", "Monty Python")

# Imprime el resultado de la búsqueda de la primera ocurrencia
print(a)

# Imprime el resultado de encontrar todas las ocurrencias
print(b)
```

```
<re.Match object; span=(4, 5), match='y'>
['y', 'y']
```

Explicación del Código

Este código utiliza la biblioteca `re` de Python para realizar operaciones de búsqueda de patrones en cadenas de texto.

1. **Importación de la biblioteca `re`**:
   - `import re`: Esto importa el módulo de expresiones regulares, que permite trabajar con patrones de texto en Python.
2. **Uso de `re.search`**:
   - `a = re.search("y", "Monty Python")`:
     - Esta función busca la primera ocurrencia de la letra "y" en la cadena "Monty Python".
     - Si se encuentra el patrón, `re.search` devuelve un objeto de coincidencia que contiene información sobre la ubicación de la coincidencia; de lo contrario, devuelve `None`.
3. **Uso de `re.findall`**:
   - `b = re.findall("y", "Monty Python")`:
     - Esta función busca todas las ocurrencias de la letra "y" en la cadena "Monty Python".
     - Devuelve una lista que contiene todas las coincidencias encontradas; si no se encuentra el patrón, devuelve una lista vacía.
4. **Impresión de resultados**:
   - `print(a)`:
     - Imprime el resultado de `re.search`. Si "y" está presente, mostrará un objeto de coincidencia; si no, mostrará `None`.
   - `print(b)`:
     - Imprime la lista de todas las ocurrencias de "y". Si no hay coincidencias, mostrará una lista vacía.

Metacaracteres

De por sí tienen otros significados (que enseguida veremos). Si queremos usarlos de forma literal hay que *escaparlos*. Son los siguientes:

```
\ ^ $ . | ? * + ( ) [ ] { }
```

```
# Busca la cadena "¿Qué es una almáciga?" en el texto dado
a = re.search("¿Qué es una almáciga\?", "¿Qué es una almáciga?")

# Imprime el resultado de la búsqueda
print(a)  # Muestra un objeto de coincidencia si se encuentra la
cadena; de lo contrario, None.
```

```
<re.Match object; span=(0, 21), match='¿Qué es una almáciga?'>
```

Explicación del Código

Este código utiliza la biblioteca `re` de Python para buscar un patrón específico dentro de una cadena de texto.

1. **Importación de la biblioteca `re`**:
   - `import re`: Esto importa el módulo de expresiones regulares, que permite realizar operaciones de búsqueda y manipulación de texto basadas en patrones.
2. **Uso de `re.search`**:
   - `a = re.search("¿Qué es una almáciga\?", "¿Qué es una almáciga?")`:
     - Esta función busca la cadena exacta "¿Qué es una almáciga?" en el texto proporcionado.
     - El carácter de escape `\?` se utiliza para indicar que el signo de interrogación es parte de la cadena que se busca, no un operador especial.
     - Si se encuentra el patrón, `re.search` devuelve un objeto de coincidencia que contiene información sobre la ubicación de la coincidencia; de lo contrario, devuelve `None`.
3. **Impresión de resultados**:
   - `print(a)`:
     - Imprime el resultado de `re.search`. Si la cadena "¿Qué es una almáciga?" está presente, mostrará un objeto de coincidencia que incluye la posición de la coincidencia en la cadena; si no se encuentra, mostrará `None`.

Hay caracteres literales que pueden escaparse para darles un uso diferente; también lo vamos a ver.

Una nota en la documentación de re sobre el uso de la barra para escapar y la `r` antes de las comillas.

```
# Busca todas las coincidencias del patrón (cadena vacía) en el texto
dado
a = re.findall("", "Todo el mundo sabe que 1+1=2")

# Imprime el resultado de la búsqueda
print(a)  # Muestra una lista con todas las coincidencias encontradas
```

```
['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '',
'', '', '', '', '', '', '', '', '', '', '', '']
```

Explicación del Código

Este código utiliza la biblioteca `re` de Python para buscar un patrón específico dentro de una cadena de texto.

1. **Importación de la biblioteca `re`**:
   - `import re`: Esto importa el módulo de expresiones regulares, que permite realizar operaciones de búsqueda y manipulación de texto basadas en patrones.
2. **Uso de `re.findall`**:
   - `a = re.findall("", "Todo el mundo sabe que 1+1=2")`:
     - Esta función busca todas las coincidencias del patrón proporcionado (en este caso, una cadena vacía `""`) en el texto dado.
     - Dado que una cadena vacía coincide entre cada carácter de la cadena de texto (incluyendo antes y después de cada carácter), esta función devolverá una lista que contiene múltiples coincidencias.
3. **Impresión de resultados**:
   - `print(a)`:
     - Imprime el resultado de `re.findall`. En este caso, mostrará una lista que contiene un número de elementos igual a la longitud de la cadena de texto más uno.

Ejercicio

Para encontrar `1+1=2`, ¿qué regex hay que usar?

```python
# Cadena de texto donde buscamos la expresión
text = "Todo el mundo sabe que 1+1=2"

# Definir el patrón
pattern = r'1\+1=2'

# Buscar la expresión en el texto
match = re.search(pattern, text)

# Comprobar si se encontró una coincidencia
if match:
    print("Se encontró la expresión:", match.group())
else:
    print("No se encontró la expresión.")
```

```
Se encontró la expresión: 1+1=2
```

Explicación del Código

Este código busca una expresión regular específica en una cadena de texto.

1. **Definición del patrón y la cadena de texto**:
   - `pattern = r'1\+1=2'`:
     - Se define una expresión regular que busca la secuencia literal `1+1=2`.
     - El símbolo `+` es escapado con una barra invertida (`\`) para que sea tratado como un carácter literal, ya que en expresiones regulares tiene un significado especial (uno o más).
2. **Uso de `search`**:
   - `a = search(pattern, "Todo el mundo sabe que 1+1=2")`:
     - Esta función busca la primera coincidencia del patrón definido en la cadena de texto proporcionada.
     - Si encuentra una coincidencia, devuelve un objeto de coincidencia; si no, devuelve `None`.
3. **Impresión de resultados**:
   - `print(a)`:
     - Imprime el resultado de la búsqueda.
     - Si la coincidencia fue encontrada, el objeto de coincidencia se imprimirá (esto incluirá información sobre la posición de la coincidencia). Si no se encontró nada, imprimirá `None`.

Sets y rangos de caracteres

Con los corchetes cuadrados `[]` creamos un set o conjunto, es decir, buscamos un carácter de entre los que metamos dentro de los corchetes.

```
a = re.findall("gui[oó]n", "Antes escribíamos guión y ahora guion")
print(a)
```

```
['guión', 'guion']
```

Explicación del Código

Este código utiliza una expresión regular para buscar coincidencias de una palabra específica en una cadena de texto, donde permite variaciones en uno de los caracteres.

1. **Definición del patrón**:
   - `pattern = "gui[oó]n"`:
     - Aquí se define una expresión regular que busca la palabra "guion" o "guión".
     - El conjunto `[oó]` permite que se coincida con `o` o `ó`, haciendo que la búsqueda sea flexible en cuanto a la letra que se use en esa posición.
2. **Uso de `re.findall`**:
   - `a = re.findall(pattern, "Antes escribíamos guión y ahora guion")`:
     - La función `findall` busca todas las coincidencias del patrón en la cadena de texto proporcionada.
     - Devuelve una lista que contiene todas las coincidencias encontradas.

3. **Impresión de resultados**:
   - `print(a)`:
     - Imprime la lista de coincidencias que se encontraron en la cadena de texto.
     - En este caso, debería mostrar `['guión', 'guion']`, ya que ambas variaciones de la palabra están presentes en el texto.

Si junto a los corchetes usamos el guion `-`, indicamos un rango. Esto es muy útil para capturar, por ejemplo, rangos de números, letras y más.

```
indice = """1. Prólogo
2. Introducción
3. Aspectos clave"""
a = re.findall("[0-9]\. ", indice)
print(a)
```

```
['1. ', '2. ', '3. ']
```

Explicación del Código

Este código utiliza una expresión regular para buscar patrones específicos en un texto que contiene una lista numerada.

1. **Definición del índice**:
   - `indice = """1. Prólogo\n2. Introducción\n3. Aspectos clave"""`:
     - Aquí se define una cadena multilinea que representa un índice con números seguidos de un punto y un espacio, que preceden a los títulos.
2. **Definición del patrón**:
   - `pattern = "[0-9]\. "`:
     - Esta expresión regular busca un dígito del 0 al 9 (`[0-9]`), seguido de un punto (`\.`) y un espacio.
     - El uso de `-` dentro de los corchetes en este caso indica un rango, permitiendo capturar cualquier número del 0 al 9.
3. **Uso de `re.findall`**:
   - `a = re.findall(pattern, indice)`:
     - La función `findall` busca todas las coincidencias del patrón en la cadena `indice`.
     - Devuelve una lista que contiene todas las coincidencias encontradas.
4. **Impresión de resultados**:
   - `print(a)`:
     - Imprime la lista de coincidencias que se encontraron en el índice.
     - En este caso, debería mostrar `['1. ', '2. ', '3. ']`, ya que se han encontrado todos los números seguidos de un punto y un espacio en la cadena.

Se puede usar más de un rango a la vez en un mismo set; por ejemplo, podemos hacer que `[a-z]` sea *case-insensitive* poniendo al lado el rango de letras en mayúsculas: `[a-zA-Z]`. Esto permite capturar tanto letras minúsculas como mayúsculas en una misma expresión.

Los corchetes también nos sirven, junto con el acento circunflejo `^`, para negar caracteres:

```
a = re.findall("q[^u]", "qué quién qé qién")
print(a)
```

```
['qé', 'qi']
```

Uso de rangos múltiples y negación de caracteres

Se puede usar más de un rango a la vez en un mismo set; por ejemplo, podemos hacer que `[a-z]` sea *case-insensitive* poniendo al lado el rango de letras en mayúsculas: `[a-zA-Z]`. Esto permite capturar tanto letras minúsculas como mayúsculas en una misma expresión.

Explicación del Código

En este código, se utiliza una expresión regular para buscar patrones específicos en una cadena que contiene varias palabras.

1. **Uso de `re.findall`**:
   - `a = re.findall("q[^u]", "qué quién qé qién")`:
     - Aquí, la expresión regular busca la letra "q" seguida de un carácter que no sea "u".
     - El `[^u]` indica que se está negando el carácter "u", lo que significa que la búsqueda encontrará cualquier carácter que siga a "q" que no sea "u".
2. **Resultado de la búsqueda**:
   - El método `findall` devolverá una lista con todas las coincidencias que cumplen con el patrón especificado.
   - En este caso, la búsqueda se realizará sobre la cadena `"qué quién qé qién"`.
3. **Impresión de resultados**:
   - `print(a)`:
     - Imprime la lista de coincidencias encontradas.
     - En este caso, la salida será `['qé', 'qi']`, que representa las instancias de "q" que están seguidas de un carácter que no es "u" (en este caso, "é" y "i").

Si metemos más caracteres dentro de los corchetes, niega todos:

```
a = re.findall("q[^ui]", "qué quién qé qién")
print(a)
```

```
['qé']
```

Explicación del Código

En este código, se utiliza una expresión regular para buscar patrones específicos en una cadena que contiene varias palabras.

1. **Uso de `re.findall`**:
   - `a = re.findall("q[^ui]", "qué quién qé qién")`:
     - La expresión regular busca la letra "q" seguida de un carácter que no sea "u" ni "i".
     - El `[^ui]` indica que se está negando tanto el carácter "u" como el carácter "i", lo que significa que la búsqueda encontrará cualquier carácter que siga a "q" que no sea "u" ni "i".
2. **Resultado de la búsqueda**:
   - El método `findall` devolverá una lista con todas las coincidencias que cumplen con el patrón especificado.
   - En este caso, la búsqueda se realizará sobre la cadena `"qué quién qé qién"`.
3. **Impresión de resultados**:
   - `print(a)`:
     - Imprime la lista de coincidencias encontradas.
     - En este caso, la salida será `['qé']`, que representa la instancia de "q" que está seguida de "é", ya que "u" y "i" están excluidos.

```
texto = """de ninguna manera-dijo
“Señores”—comenzó—”no vamos a…"""
a = re.findall("", texto)
print(a)
```

```
['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '',
'', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '',
'', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '',
'', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '',
'', '', '']
```

Explicación del Código

Este código utiliza la biblioteca de expresiones regulares de Python para buscar un patrón específico dentro de una cadena de texto.

1. **Definición de la cadena de texto**:
   - `texto = """de ninguna manera-dijo “Señores”—comenzó—”no vamos a…""`:
     - Aquí se define un bloque de texto que contiene varias líneas y caracteres especiales como guiones y comillas.
2. **Uso de `re.findall`**:
   - `a = re.findall("", texto)`:
     - La función `re.findall` se utiliza para buscar todas las coincidencias del patrón especificado (en este caso, una cadena vacía `""`) en el texto dado.

    - Una cadena vacía coincide entre cada carácter de la cadena de texto (incluyendo antes y después de cada carácter), por lo que se generará una lista que contiene múltiples coincidencias.
3. **Impresión de resultados**:
    - `print(a)`:
        - Imprime el resultado de `re.findall`.
        - En este caso, mostrará una lista que contiene un número de elementos igual a la longitud de la cadena de texto más uno.
        - Por ejemplo, si el texto tiene 100 caracteres, la salida será una lista de 101 elementos vacíos.

Ejercicio

Para encontrar `a-d`, `”–c` y `ó–”`, ¿qué regex hay que usar? Ojo, que no es lo mismo el guion `-` que la raya `–`.

```
# Texto de ejemplo modificado
texto = """En la reunión se mencionó a-d,
y también se dijo que había un caso de “acción”–c
en el informe. Además, se vio ó–”
en la presentación final. Es importante no olvidar a-b
porque también es parte del análisis."""

# Patrón regex para encontrar las coincidencias
pattern = r'a-d|”–c|ó–”'

# Buscar las coincidencias en el texto
resultados = re.findall(pattern, texto)

# Imprimir los resultados
print(resultados)
```

```
['a-d', '”–c', 'ó–”']
```

Explicación del Código

Este código utiliza la biblioteca de expresiones regulares de Python para buscar un patrón específico dentro de una cadena de texto.

1. **Definición de la cadena de texto**:
    - `texto = """En la reunión se mencionó a-d,  y también se dijo que había un caso de “acción”–c  en el informe. Además, se vio ó–”  en la presentación final. Es importante no olvidar a-b  porque también es parte del análisis."""`:
        - Aquí se define un bloque de texto que contiene varias líneas y caracteres especiales como guiones y comillas. Este texto es el que se utilizará para buscar coincidencias del patrón.
2. **Definición del patrón regex**:
    - `pattern = r'a-d|”–c|ó–”'`:

- Se establece un patrón de búsqueda que consiste en tres partes separadas por el operador `|` (or):
  - `a-d`: busca la secuencia de caracteres "a-d".
  - `”–c`: busca la secuencia que incluye “”—c.
  - `ó–”`: busca la secuencia que incluye “ó—”.
- Este patrón permite identificar cualquiera de estas secuencias en el texto.

3. **Uso de `re.findall`**:
   - `resultados = re.findall(pattern, texto)`:
     - La función `re.findall` se utiliza para buscar todas las coincidencias del patrón especificado en el texto dado.
     - Devuelve una lista que contiene todas las instancias que coinciden con el patrón.
4. **Impresión de resultados**:
   - `print(resultados)`:
     - Imprime la lista de coincidencias encontradas.
     - Mostrará todas las secuencias que coinciden con el patrón especificado.
     - Por ejemplo, la salida puede ser `['a-d', '”–c', 'ó–”']`, lo que indica que se encontraron correctamente las tres secuencias en el texto original.

Repetición y alternancia

Con `?` indicamos que el carácter anterior es opcional.

```
a = re.findall("amigos?", "¿Tienes un amigo o tienes muchos amigos?")
print(a)

['amigo', 'amigos']
```

Explicación del Código

Este código utiliza la biblioteca de expresiones regulares de Python para buscar un patrón específico dentro de una cadena de texto.

1. **Uso de `re.findall`**:
   - `a = re.findall("amigos?", "¿Tienes un amigo o tienes muchos amigos?")`:
     - La función `re.findall` se utiliza para buscar todas las coincidencias del patrón especificado en el texto dado.
     - El patrón `amigos?` busca la palabra "amigo" y también "amigos".
     - El símbolo `?` indica que la letra 's' es opcional, lo que significa que se coincidirá con "amigo" (sin 's') o "amigos" (con 's').
2. **Impresión de resultados**:
   - `print(a)`:
     - Imprime la lista de coincidencias encontradas.

- En este caso, el resultado será una lista que contiene las palabras que coincidieron con el patrón.
- Por ejemplo, la salida será `['amigos', 'amigo']`, lo que indica que se encontraron ambas variaciones en el texto original.

Con `+` buscamos que el carácter anterior salga una o más veces.

```
# Capturar cualquier conjunto de dos o más espacios
a = re.sub(" +", " ", "Ciudad del  Cabo, una ciudad   de 3,7 millones
de  habitantes")
print(a)

Ciudad del Cabo, una ciudad de 3,7 millones de habitantes
```

Explicación del Código

Este código utiliza la biblioteca de expresiones regulares de Python para reemplazar múltiples espacios en una cadena de texto con un solo espacio.

1. **Uso de `re.sub`**:
   - `a = re.sub(" +", " ", "Ciudad del  Cabo, una ciudad   de 3,7 millones de  habitantes")`:
     - La función `re.sub` se utiliza para buscar un patrón en el texto y reemplazarlo con una cadena especificada.
     - El patrón `" +"` busca uno o más espacios en blanco en la cadena.
       - Aquí, el símbolo `+` indica que debe coincidir con una o más ocurrencias del carácter anterior (en este caso, el espacio).
     - El texto original es `"Ciudad del  Cabo, una ciudad   de 3,7 millones de  habitantes"`, que contiene múltiples espacios entre palabras.
     - El segundo argumento, `" "`, indica que todos los espacios encontrados serán reemplazados por un solo espacio.
2. **Impresión de resultados**:
   - `print(a)`:
     - Imprime la cadena resultante después de realizar la sustitución.
     - En este caso, la salida será `"Ciudad del Cabo, una ciudad de 3,7 millones de habitantes"`, donde todos los grupos de dos o más espacios han sido reducidos a un solo espacio.

El asterisco `*` se puede entender como una mezcla de `?` y `+`: indicamos que es opcional, pero que, si sale, lo haga una o más veces.

```
# Capturar cualquier palabra entrecomillada
a = re.findall('[«"`][a-zA-Z ]*[´"»]', '''¿Cuántas veces dicen «ni»
los caballeros
               que dicen "ni" en `Monty Python y los caballeros de la
mesa cuadrada´?''')
print(a)
```

```
['«ni»', '"ni"', '`Monty Python y los caballeros de la mesa cuadrada ´']
```

Explicación del Código

Este código utiliza la biblioteca de expresiones regulares de Python para buscar palabras que están entre comillas o caracteres específicos en una cadena de texto.

1. **Uso de `re.findall`**:
   - `a = re.findall('[«"][a-zA-Z ]*[´"»]', '¿Cuántas veces dicen «ni» los caballeros que dicen "ni" en Monty Python y los caballeros de la mesa cuadrada´?''')`:
     - La función `re.findall` se utiliza para buscar todas las coincidencias del patrón especificado en el texto dado.
     - El patrón `['«"][a-zA-Z ]*[´"»]` se desglosa de la siguiente manera:
       - `['«"]`: Coincide con cualquier carácter de apertura, que puede ser una comilla simple (`'`), comilla doble (`"`), comillas angulares (`«`), o acentos graves (`` ` ``).
       - `[a-zA-Z ]*`: Coincide con cero o más caracteres alfabéticos (tanto mayúsculas como minúsculas) y espacios. Esto permite capturar palabras compuestas por letras y espacios.
       - `[´"»]`: Coincide con cualquier carácter de cierre, que puede ser una comilla simple invertida (`´`), comilla doble (`"`), o comillas angulares de cierre (`»`).
     - El texto original es un bloque de texto que incluye varias frases, algunas de las cuales contienen palabras entre comillas.
2. **Impresión de resultados**:
   - `print(a)`:
     - Imprime la lista de palabras que han sido capturadas por el patrón de búsqueda.
     - En este caso, la salida será `['«ni»', '"ni"', '´Monty Python y los caballeros de la mesa cuadrada´']`, que incluye todas las palabras que están entre comillas en el texto original.

Podemos indicar el número exacto de apariciones de caracteres que nos interesan metiéndolo entre las llaves `{}`, o un rango si separamos los números mediante comas.

```python
# Capturar cualquier número entre 1000 y 9999
a = re.findall(" [1-9][0-9]{3},", "125, 987, 2940, 5982, 13943, 38492, 748392, 404921")
print(a)
# Capturar cualquier número entre 100 y 99999
a = re.findall(" [1-9][0-9]{2,4},", "125, 987, 2940, 5982, 13943, 38492, 748392, 404921")
print(a)
```

```
[' 2940,', ' 5982,']
[' 987,', ' 2940,', ' 5982,', ' 13943,', ' 38492,']
```

Explicación del Código

Este código utiliza la biblioteca de expresiones regulares de Python para capturar números dentro de rangos específicos en una cadena de texto.

1. **Capturar números entre 1000 y 9999**:
   - `a = re.findall(" [1-9][0-9]{3},", "125, 987, 2940, 5982, 13943, 38492, 748392, 404921")`:
     - La función `re.findall` busca todas las coincidencias del patrón especificado en el texto dado.
     - El patrón `" [1-9][0-9]{3},"` se desglosa de la siguiente manera:
       - `" "`: Un espacio antes del número, que asegura que estamos capturando números que no están al principio de la cadena.
       - `[1-9]`: El primer dígito debe ser un número del 1 al 9, evitando que el número comience con cero.
       - `[0-9]{3}`: Debe haber exactamente tres dígitos que pueden ser del 0 al 9, formando así un total de cuatro dígitos.
       - `","`: El número debe estar seguido de una coma, indicando que es parte de una lista.
     - En el texto proporcionado, los números que cumplen con estas condiciones son `2940` y `5982`, que son parte de la salida `[' 2940,', ' 5982,']`.
2. **Impresión de resultados**:
   - `print(a)`:
     - Imprime la lista de números que han sido capturados por el patrón de búsqueda.
     - La salida es `[' 2940,', ' 5982,']`, que incluye todos los números entre 1000 y 9999 en la cadena.
3. **Capturar números entre 100 y 99999**:
   - `a = re.findall(" [1-9][0-9]{2,4},", "125, 987, 2940, 5982, 13943, 38492, 748392, 404921")`:
     - Esta vez, el patrón `" [1-9][0-9]{2,4},"` permite capturar números de tres a cinco dígitos.
     - Se descompone de la siguiente manera:
       - `" "`: Un espacio antes del número.
       - `[1-9]`: El primer dígito debe ser un número del 1 al 9.
       - `[0-9]{2,4}`: Debe haber entre dos y cuatro dígitos adicionales, lo que permite capturar números de tres a cinco dígitos en total.
       - `","`: El número debe estar seguido de una coma.
     - En el texto proporcionado, los números que cumplen con estas condiciones son `125`, `987`, `2940`, `5982`, `13943`, y `38492`, que son parte de

la salida `[' 987,', ' 2940,', ' 5982,', ' 13943,', ' 38492,']`.

4. **Impresión de resultados**:
   - `print(a)`:
     - Imprime la lista de números que han sido capturados por este nuevo patrón de búsqueda.
     - La salida es `[' 987,', ' 2940,', ' 5982,', ' 13943,', ' 38492,']`, que incluye todos los números entre 100 y 99999 en la cadena.

La pleca `|` permite la alternancia entre dos opciones:

```
a = re.findall("Monty Python|the Pythons", """Monty Python and the
Holy Grail was based on Arthurian legend
                and was directed by Jones and Gilliam. Again, the
latter also contributed linking animations
                (and put together the opening credits). Along with the
rest of the Pythons, Jones and Gilliam
                performed several roles in the film, but Chapman took
the lead as King Arthur. Cleese returned
                to the group for the film, feeling that they were once
again breaking new ground. Holy Grail
                was filmed on location, in picturesque rural areas of
Scotland, with a budget of only £229,000;
                the money was raised in part with investments from
rock groups such as Pink Floyd, Jethro Tull,
                and Led Zeppelin—and UK music industry entrepreneur
Tony Stratton-Smith (founder and owner of
                the Charisma Records label, for which the Pythons
recorded their comedy albums).""")
print(a)
```

```
['Monty Python', 'the Pythons', 'the Pythons']
```

Explicación del Código

Este código utiliza la biblioteca de expresiones regulares de Python para buscar coincidencias de frases específicas en un bloque de texto.

1. **Definición de la cadena de texto**:
   - `"""Monty Python and the Holy Grail was based on Arthurian legend ... """`:
     - Aquí se define un bloque de texto que contiene información sobre Monty Python y su obra "Monty Python and the Holy Grail". Este texto incluye varios detalles sobre la producción y los miembros del grupo.
2. **Uso de `re.findall`**:
   - `a = re.findall("Monty Python|the Pythons", ...)`:

- La función `re.findall` se utiliza para buscar todas las coincidencias del patrón especificado en el texto dado.
- El patrón `Monty Python|the Pythons` busca las ocurrencias de las frases "Monty Python" o "the Pythons" en el texto.
  - `Monty Python`: Coincide exactamente con la cadena "Monty Python".
  - `the Pythons`: Coincide exactamente con la cadena "the Pythons".
- La función devolverá una lista con todas las coincidencias encontradas.

3. **Impresión de resultados**:
   - `print(a)`:
     - Imprime el resultado de `re.findall`.
     - En este caso, la salida será una lista que contiene las coincidencias encontradas en el texto original.
     - Por ejemplo, en el texto proporcionado, la salida fue `['Monty Python', 'the Pythons', 'the Pythons']`, indicando que "Monty Python" apareció una vez y "the Pythons" apareció dos veces.
4. **Salida esperada**:
   - La lista impresa refleja todas las instancias de "Monty Python" y "the Pythons" encontradas en el texto, mostrando que se ha cumplido el objetivo de la búsqueda.

Ejercicios

Escribamos un traductor balleno-castellano, que sustituya todas las vocales repetidas por una sola vocal. Rellena con regex:

```
saludo = "hooooooooolaaaaaa, señooooooora balleeeeeeeeenaaaaaaaaa"
saludo = re.sub("", "a", saludo)
saludo = re.sub("", "e", saludo)
saludo = re.sub("", "i", saludo)
saludo = re.sub("", "o", saludo)
saludo = re.sub("", "u", saludo)
print(saludo)
```

```
uouiuoueuouiuouauouiuoueuouiuouhuouiuoueuouiuouauouiuoueuouiuouououiuo
ueuouiuouauouiuoueuouiuouououiuoueuouiuouauouiuoueuouiuouououiuoueuoui
uouauouiuoueuouiuouououiuoueuouiuouauouiuoueuouiuouououiuoueuouiuouauo
uiuoueuouiuouououiuoueuouiuouauouiuoueuouiuouououiuoueuouiuouauouiuoue
uouiuouououiuoueuouiuouauouiuoueuouiuouououiuoueuouiuouauouiuoueuouiuo
uououiuoueuouiuouauouiuoueuouiuouluouiuoueuouiuouauouiuoueuouiuouauoui
uoueuouiuouauouiuoueuouiuouauouiuoueuouiuouauouiuoueuouiuouauouiuoueuo
uiuouauouiuoueuouiuouauouiuoueuouiuouauouiuoueuouiuouauouiuoueuouiuoua
uouiuoueuouiuouauouiuoueuouiuouauouiuoueuouiuou,uouiuoueuouiuouauouiuo
ueuouiuou
uouiuoueuouiuouauouiuoueuouiuousuouiuoueuouiuouauouiuoueuouiuoueuouiuo
ueuouiuouauouiuoueuouiuouñuouiuoueuouiuouauouiuoueuouiuouououiuoueuoui
uouauouiuoueuouiuouououiuoueuouiuouauouiuoueuouiuouououiuoueuouiuouauo
```

```
uiuoueuouiuouououiuoueuouiuouauouiuoueuouiuouououiuoueuouiuouauouiuoue
uouiuouououiuoueuouiuouauouiuoueuouiuouououiuoueuouiuouauouiuoueuouiuo
uououiuoueuouiuouauouiuoueuouiuouruouiuoueuouiuouauouiuoueuouiuouauoui
uoueuouiuouauouiuoueuouiuou
uouiuoueuouiuouauouiuoueuouiuoubuouiuoueuouiuouauouiuoueuouiuouauouiuo
ueuouiuouauouiuoueuouiuouluouiuoueuouiuouauouiuoueuouiuouluouiuoueuoui
uouauouiuoueuouiuoueuouiuoueuouiuouauouiuoueuouiuoueuouiuoueuouiuouauo
uiuoueuouiuoueuouiuoueuouiuouauouiuoueuouiuoueuouiuoueuouiuouauouiuoue
uouiuoueuouiuoueuouiuouauouiuoueuouiuoueuouiuoueuouiuouauouiuoueuouiuo
ueuouiuoueuouiuouauouiuoueuouiuoueuouiuoueuouiuouauouiuoueuouiuoueuoui
uoueuouiuouauouiuoueuouiuounuouiuoueuouiuouauouiuoueuouiuouauouiuoueuo
uiuouauouiuoueuouiuouauouiuoueuouiuouauouiuoueuouiuouauouiuoueuouiuoua
uouiuoueuouiuouauouiuoueuouiuouauouiuoueuouiuouauouiuoueuouiuouauouiuo
ueuouiuouauouiuoueuouiuouauouiuoueuouiuouauouiuoueuouiuouauouiuoueuoui
uouauouiuoueuouiuouauouiuoueuouiuouauouiuoueuouiuouauouiuoueuouiuou
```

Explicación del Código

Este código utiliza la biblioteca de expresiones regulares de Python para modificar una cadena de texto inicial.

1. **Definición de la cadena de saludo**:
   - `saludo = "hooooooooooolaaaaaa, señooooooooora balleeeeeeeeenaaaaaaaaa"`:
     - Aquí se define una cadena de texto que contiene un saludo. Este saludo incluye muchas letras repetidas, lo que lo hace interesante para la manipulación.
2. **Uso de `re.sub`**:
   - Las siguientes líneas utilizan la función `re.sub` para sustituir todas las coincidencias del patrón proporcionado (en este caso, una cadena vacía `""`) por las vocales "a", "e", "i", "o", "u":
     - `saludo = re.sub("", "a", saludo)`:
       - Sustituye cada posición en la cadena (incluyendo entre cada carácter y al principio y al final) por la letra "a". Esto produce una duplicación de los caracteres existentes en la cadena y la inserción de "a" en cada espacio posible.
     - `saludo = re.sub("", "e", saludo)`:
       - Lo mismo, pero reemplazando por la letra "e".
     - `saludo = re.sub("", "i", saludo)`:
       - Reemplaza por "i".
     - `saludo = re.sub("", "o", saludo)`:
       - Reemplaza por "o".
     - `saludo = re.sub("", "u", saludo)`:
       - Reemplaza por "u".
3. **Impresión de resultados**:
   - `print(saludo)`:

    - Imprime la cadena `saludo` después de haber sido modificada por las sustituciones anteriores.
    - Debido a que la función `re.sub` se aplica a una cadena vacía, cada carácter en la cadena original se separa y se añaden las vocales entre ellos, lo que resultará en una cadena muy larga con muchas vocales.
4. **Salida esperada**:
    - La salida mostrará una cadena que consiste en los caracteres originales del saludo intercalados con múltiples vocales "a", "e", "i", "o" y "u".
    - Por ejemplo, la cadena final será muy extensa y visualmente impactante, ya que cada letra original se habrá duplicado y mezclado con las vocales.

Consideraciones

- El uso de `re.sub("", ...)` para una cadena vacía es poco convencional, ya que generalmente se utiliza para patrones específicos. Este enfoque produce resultados interesantes al modificar el contenido original de la cadena.

```
a = re.findall("", "#63ffed #daffbb #ff787b")
print(a)

['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '']
```

Explicación del Código

Este código utiliza la biblioteca de expresiones regulares de Python para buscar un patrón específico dentro de una cadena de texto que contiene códigos de colores en formato hexadecimal.

1. **Definición de la cadena de texto**:
    - `a = re.findall("", "#63ffed #daffbb #ff787b")`:
        - La función `re.findall` busca todas las coincidencias del patrón especificado en el texto dado.
        - En este caso, se proporciona una cadena vacía `""` como patrón. Esto significa que la función buscará posiciones entre cada carácter de la cadena de texto, lo que incluye espacios, antes y después de cada carácter.
2. **Impresión de resultados**:
    - `print(a)`:
        - Imprime el resultado de `re.findall`.
        - Debido a que el patrón es una cadena vacía, la salida será una lista que contiene un número de elementos igual a la longitud de la cadena de texto más uno.
        - Por ejemplo, si la cadena de texto tiene 24 caracteres (incluyendo los espacios), la salida será una lista con 25 elementos vacíos.

Salida Esperada

- La salida mostrará una lista con un número de elementos vacíos, lo que resulta de la búsqueda de coincidencias de una cadena vacía en el texto. La salida se verá así:

Ejercicio

Para encontrar códigos RGB de colores, ¿qué regex tendríamos que escribir?

```
texto = "#63ffed #daffbb #ff787b y otros colores como #abc y #123456."
patron = r'#[0-9a-fA-F]{6}|#[0-9a-fA-F]{3}'

# Buscar los códigos de color en el texto
resultados = re.findall(patron, texto)

# Imprimir los resultados
print(resultados)
```

```
['#63ffed', '#daffbb', '#ff787b', '#abc', '#123456']
```

Explicación del Código

Este código utiliza la biblioteca de expresiones regulares de Python para buscar códigos de colores en formato hexadecimal dentro de una cadena de texto.

1. **Definición de la cadena de texto**:
   - `texto = "#63ffed #daffbb #ff787b y otros colores como #abc y #123456."`:
     - Aquí se define una cadena de texto que contiene varios códigos de color en formato hexadecimal, que pueden ser de 3 o 6 dígitos.
2. **Definición del patrón regex**:
   - `patron = r'#[0-9a-fA-F]{6}|#[0-9a-fA-F]{3}'`:
     - Este patrón busca:
       - `#[0-9a-fA-F]{6}`: Un símbolo de número `#` seguido de exactamente seis caracteres que pueden ser dígitos del 0 al 9 o letras de la a a la f (en minúsculas o mayúsculas). Esto representa un color en formato hexadecimal de 6 dígitos.
       - `|`: Un operador "o" que permite buscar otra opción en el patrón.
       - `#[0-9a-fA-F]{3}`: Un símbolo de número `#` seguido de exactamente tres caracteres que pueden ser dígitos del 0 al 9 o letras de la a a la f. Esto representa un color en formato hexadecimal de 3 dígitos.
3. **Uso de `re.findall`**:
   - `resultados = re.findall(patron, texto)`:
     - La función `re.findall` busca todas las coincidencias del patrón especificado en la cadena de texto dada.
     - Devuelve una lista que contiene todos los códigos de color que coinciden con el patrón.

4. **Impresión de resultados**:
   - `print(resultados)`:
     - Imprime el resultado de la búsqueda, que será una lista de los códigos de color encontrados en el texto.

Salida Esperada

- La salida mostrará una lista con todos los códigos de color que han sido encontrados en la cadena de texto. Para el texto proporcionado, la salida será:

Comodines

Ciertos caracteres literales se pueden escapar, como hacíamos con los metacaracteres, para usos especiales. Tienen la particularidad de que al ponerlos en mayúsculas, los negamos.

- Con `\d` encontramos dígitos: `\d` = `[0-9]`; `\D` = `[^0-9]`
- Con `\w` encontramos caracteres alfanuméricos y la barra baja: `\w` = `[a-zA-Z0-9_]`; `\W` = `[^a-zA-Z0-9_]`

Caracteres invisibles

- Con `\t` encontramos el tabulador.
- Con `\n` encontramos el carácter de nueva línea.
- Con `\r` encontramos el carácter de retorno de carro.
- Con `\s` encontramos espacios, tabuladores y saltos de línea: `\s` = `[ \t\r\n]`; `\S` = `[^ \t\r\n]`

En Windows, por defecto, al crear un nuevo párrafo en los programas de procesamiento de texto, en realidad se están imprimiendo `\r` y `\n`. En Linux, se imprime solo `\n`.

El punto

El punto `.` encuentra casi todo: todo menos precisamente los saltos de línea (aunque esto es configurable). ¡Hay que tener mucho cuidado con el punto!

Ejercicio

Tenemos una lista de términos sacados de un diccionario médico y queremos deshacernos de prefijos y sufijos. ¿Qué regex hay que usar?

```
# Lista de términos médicos
entries = [
    "acantocéfalo, la",
    "acéfalo, la",
    "bicéfalo, la",
    "braquicéfalo, la",
    "bucéfalo",
    "calocéfalo, la",
    "céfalo",
    "cefalo-, -céfalo, la",
```

```
    "cinocéfalo",
    "dolicocéfalo, la",
    "encéfalo",
    "hidrocéfalo, la",
    "macrocéfalo, la",
    "mesocéfalo, la",
    "microcéfalo, la",
    "policéfalo, la",
    "termocéfalo, la",
    "tricéfalo, la",
    "estomatitis",
    "faringitis",
    "fascitis",
    "flebitis",
    "flojeritis",
    "gastritis",
    "gastroenteritis",
    "gingivitis",
    "glositis",
    "hepatitis",
    "iritis",
    "laringitis",
    "linfangitis",
    "litis",
    "mastitis",
    "meningitis",
    "neurocirugía, la",
    "psicopatía, la",
    "hipertensión, la",
    "hipotermia, la",
    "poliartritis, la",
    "macroadenoma, el",
    "microangiopatía, la",
    "cardiopatía, la",
    "hepatomegalia, la",
    "neumonitis, la",
    "cistitis, la",
    "otitis, la",
    "sinusitis, la",
    "tendinitis, la",
    "artrosis, la",
    "osteoporosis, la",
    "poliomielitis, la",
    "quimioterapia, la",
    "radioterapia, la",
    "antibióticos, los",
    "anemia, la",
    "quistes, los",
    "micosis, la",
```

```
    "toxoplasmosis, la",
    "hemorragia, la"
]

# Lista para términos limpios
entries_clean = []

# Patrón regex mejorado con más prefijos y sufijos
pattern = r'^(?:a-|acanto|anti-|bi|braqui|cino|doli|endo|epid|hidro|
macro|meso|micro|neo|poli|pre|sub|termo|trans)?([^,-]+)(?:-céfalo|-
itis|-algia|-patía|-escopia|-ectomía|-oma|-osis|-uria|-logía|-
plastia|-tóxico)?(?:, la|, el)?$'

for entry in entries:
    match = re.match(pattern, entry)
    if match:
        # Añadir solo el término limpio a la lista
        entries_clean.append(match.group(1).strip())

# Imprimir los términos limpios
print(entries_clean)
```

```
['céfalo', 'acéfalo', 'céfalo', 'céfalo', 'bucéfalo', 'calocéfalo',
'céfalo', 'céfalo', 'cocéfalo', 'encéfalo', 'céfalo', 'céfalo',
'céfalo', 'céfalo', 'céfalo', 'céfalo', 'tricéfalo', 'estomatitis',
'faringitis', 'fascitis', 'flebitis', 'flojeritis', 'gastritis',
'gastroenteritis', 'gingivitis', 'glositis', 'hepatitis', 'iritis',
'laringitis', 'linfangitis', 'litis', 'mastitis', 'meningitis',
'neurocirugía', 'psicopatía', 'hipertensión', 'hipotermia',
'artritis', 'adenoma', 'angiopatía', 'cardiopatía', 'hepatomegalia',
'neumonitis', 'cistitis', 'otitis', 'sinusitis', 'tendinitis',
'artrosis', 'osteoporosis', 'omielitis', 'quimioterapia',
'radioterapia', 'anemia', 'micosis', 'toxoplasmosis', 'hemorragia']
```

Explicación del Código

Este código utiliza la biblioteca de expresiones regulares de Python para limpiar una lista de términos médicos, extrayendo solo la parte relevante de cada término.

1. **Definición de la lista de términos médicos**:
   - `entries`: Esta lista contiene una serie de términos médicos que pueden incluir prefijos, sufijos y artículos. Algunos términos tienen variaciones en su forma, lo que los hace más complejos para la extracción.
2. **Inicialización de una lista vacía**:
   - `entries_clean = []`: Se crea una lista vacía que almacenará los términos limpios, es decir, aquellos términos sin prefijos, sufijos y artículos.
3. **Definición del patrón regex**:
   - `pattern`: Se define un patrón regex que busca extraer los nombres de los términos médicos relevantes. El patrón es el siguiente:

- `^`: Indica el inicio de la cadena.
- `(?:...)`: Agrupación no capturante que permite incluir varios prefijos opcionales, como `a-`, `acanto`, `anti-`, etc.
- `([^,-]+)`: Captura el término principal, que consiste en uno o más caracteres que no sean comas `,` o guiones `-`.
- `(?:...)`: Otra agrupación no capturante para los sufijos que pueden aparecer, como `-céfalo`, `-itis`, `-algia`, etc.
- `(?:, la|, el)?`: Opcionalmente captura el artículo definido que puede aparecer al final, como `, la` o `, el`.
- `$`: Indica el final de la cadena.

4. **Iteración sobre los términos**:
   - `for entry in entries:`: Se recorre cada término en la lista `entries`.
   - `match = re.match(pattern, entry)`: Se aplica el patrón regex a cada término. Si se encuentra una coincidencia, `match` contendrá el resultado.
5. **Almacenamiento de términos limpios**:
   - `if match:`: Verifica si se encontró una coincidencia.
     - `entries_clean.append(match.group(1).strip())`: Si hay una coincidencia, se agrega la parte capturada (el término limpio) a la lista `entries_clean`, usando `strip()` para eliminar cualquier espacio en blanco adicional.
6. **Impresión de resultados**:
   - `print(entries_clean)`: Finalmente, se imprime la lista de términos limpios que fueron extraídos de los términos médicos originales.

Anclas

Las anclas no se corresponden con ningún carácter, sino con posiciones. Su particularidad reside en que si lo que queremos es reemplazar una string por otra, no tenemos que ponerlos en la cadena meta.

El acento circunflejo `^` indica el principio de una línea y `$`, el final. Los límites de las palabras también los podemos encontrar, con `\b` (y usar su contrario, `\B`).

```
indice = """1. Prólogo
2. Introducción
3. Aspectos clave
4. Historia desde 1979. Un relato único"""
a = re.findall(r"^[0-9]\. ", indice, flags=re.M)
print(a)
```

```
['1. ', '2. ', '3. ', '4. ']
```

Explicación del Código

Este código utiliza la función `re.findall` para buscar números seguidos de un punto y un espacio en un índice, asegurando que se encuentren al inicio de cada línea.

1. **Definición de la cadena de texto**:
   - `indice`: Se define un bloque de texto que contiene varios elementos de un índice numerado. Los números están seguidos de un punto y un espacio, por ejemplo, "1. Prólogo" y "4. Historia desde 1979. Un relato único".
2. **Uso de `re.findall`**:
   - `a = re.findall(r"^[0-9]\. ", indice, flags=re.M)`:
     - Esta línea busca todos los números del 0 al 9 que estén al inicio de una línea (`^`), seguidos de un punto y un espacio (`\.`).
     - La opción `flags=re.M` (modo multilinea) indica que el patrón debe aplicarse al inicio de cada línea, no solo al inicio de todo el texto.
     - El patrón regex `^[0-9]\.` captura cualquier número de un solo dígito seguido de un punto y un espacio.
3. **Impresión de resultados**:
   - `print(a)`:
     - Imprime todas las coincidencias encontradas en el texto que corresponden al formato numérico con un punto y un espacio.

Ejercicio

En el texto proporcionado, las palabras con "to", "ta", "te", "tos", "tas", "tes" deben ser capturadas ya que probablemente son errores de OCR y deberían reemplazarse por "lo", "la", "le", "los", "las", "les".. ¿Qué debemos buscar?

```
import re

# Texto con errores OCR
texto = """todo estaba oscuro
era to más parecido
una de tas mejores obras
según decían, «tes asustaban»"""

# Usamos un patrón que capture "to", "ta", "te", "tos", "tas", "tes"
# \b indica el inicio o fin de palabra, t[aoe]s? captura las palabras
mencionadas
resultados = re.findall(r"\b(t[aoe]s?)\b", texto)

# Imprimir los resultados encontrados
print(resultados)
```

```
['to', 'tas', 'tes']
```

Explicación del Código

Este código utiliza la biblioteca de expresiones regulares para detectar palabras mal escritas que podrían haberse generado por errores de OCR en un texto.

1. **Definición del texto**:

- `texto = """todo estaba oscuro ... según decían, «tes asustaban»"""`:
    - El bloque de texto contiene algunas palabras mal transcritas, como "to", "tas", "tes", que se pudieron generar por un error de OCR (reconocimiento óptico de caracteres).

2. **Uso de `re.findall`**:
    - `resultados = re.findall(r"\b(t[aoe]s?)\b", texto)`:
        - El patrón `\b(t[aoe]s?)\b` busca las palabras que empiezan con "t", seguidas de "a", "o" o "e", opcionalmente seguidas de "s".
        - **Explicación del patrón**:
            - `\b`: Indica un límite de palabra, para asegurarse de que coincidimos solo con palabras completas.
            - `t[aoe]`: Busca palabras que empiecen con "t" y cuyo segundo carácter sea "a", "o" o "e".
            - `s?`: La "s" es opcional, permitiendo capturar tanto palabras en singular ("to", "ta", "te") como en plural ("tos", "tas", "tes").
            - `\b`: Otro límite de palabra para marcar el final de la palabra.
        - El patrón captura palabras como "to", "tas", "tes", entre otras.

3. **Impresión de los resultados**:
    - `print(resultados)`:
        - Imprime todas las coincidencias encontradas en el texto.

Avaricia y pereza en cuantificadores

Los cuantificadores que hemos visto antes son, por defecto, avariciosos (*greedy*); esto significa que abarcarán todo lo que puedan. En ciertas situaciones, esto puede generar resultados no deseados, como al tratar con etiquetas HTML.

Ejemplo con etiquetas HTML

Supongamos que tenemos el siguiente texto con etiquetas HTML:

```
HTML = "Podemos llamarlas <em>expresiones regulares</em>,
<em>regexp</em> o <em>regex</em>."
```

Queremos capturar todas las etiquetas HTML (...), pero si usamos la expresión regular <.+>, obtendremos un resultado inesperado.

Cuantificador avaricioso (greedy)

Primero probamos con la siguiente expresión regular:

```
a = re.findall("<.+>", HTML)
print(a)
```

```
['<em>expresiones regulares</em>, <em>regexp</em> o <em>regex</em>']
```

Podríamos intentar restringir el patrón para evitar que capture demasiado:

```
a = re.findall("<[^ ]+>", HTML)
print(a)
```

```
['<em>', '</em>', '<em>regexp</em>', '<em>regex</em>']
```

Esto tampoco funciona, ya que el patrón <[^]+> solo captura etiquetas que no contengan espacios, lo cual no es adecuado para nuestro caso, ya que las etiquetas HTML pueden contener atributos o texto más complejo.

Solución: Cuantificador perezoso (lazy)

Para solucionar esto, podemos hacer que el cuantificador sea perezoso (lazy), de modo que se detenga en la primera instancia de >. Esto se logra añadiendo el ? justo después del +, es decir, usamos <.+?>.

```
a = re.findall("<.+?>", HTML)
print(a)
```

```
['<em>', '</em>', '<em>', '</em>', '<em>', '</em>']
```

Ejercicios

Encuentra cada oración por separado en esta cita de las *Meditaciones* de Marco Aurelio:

```
# Texto de la cita
texto = "No actúes en la idea de que vas a vivir diez mil años. La necesidad ineludible pende sobre ti. Mientras vives, mientras es posible, sé virtuoso."

# Usamos una expresión regular que capture las oraciones terminadas en un punto o signo de puntuación
a = re.findall(r'[^.]+[.]+', texto)

# Imprimir las oraciones encontradas
print(a)
```

```
['No actúes en la idea de que vas a vivir diez mil años.', ' La necesidad ineludible pende sobre ti.', ' Mientras vives, mientras es posible, sé virtuoso.']
```

Explicación del Código

Este código utiliza expresiones regulares para dividir un texto en oraciones separadas.

1. **Texto de entrada**:
   - El texto es una cita que contiene varias oraciones, cada una terminada en un punto.
2. **Uso de `re.findall`**:

- La expresión regular `r'[^.]+[.]+'` se utiliza para encontrar todas las oraciones en el texto.
  - `[^.]+` coincide con uno o más caracteres que no son un punto. Esto asegura que capturemos el contenido de la oración.
  - `[.]` asegura que busquemos una o más instancias del punto que finaliza cada oración.
  - Así, la combinación de ambos patrones permite capturar cada oración completa hasta el siguiente punto.

3. **Impresión de resultados**:
   - `print(a)` imprime la lista de oraciones encontradas en el texto.
   - Como resultado, se obtendrá una lista de cadenas que representan cada oración de la cita original.

Encuentra todas las líneas aéreas:

```
# Texto con las líneas aéreas
airlines = """Andes Líneas Aéreas
Plus Ultra Líneas Aéreas
Líneas Aéreas del Estado"""

# Usamos una expresión regular para encontrar líneas aéreas
# Capturamos líneas que contengan "Líneas Aéreas" y cualquier texto
antes
a = re.findall(r'.*Líneas Aéreas.*', airlines)

# Imprimir las líneas aéreas encontradas
print(a)
```

```
['Andes Líneas Aéreas', 'Plus Ultra Líneas Aéreas', 'Líneas Aéreas del
Estado']
```

Explicación del Código

Este código utiliza expresiones regulares para identificar y extraer todas las líneas aéreas de un texto.

1. **Texto de entrada**:
   - `airlines` contiene una lista de líneas aéreas, cada una en una línea separada.
2. **Uso de `re.findall`**:
   - La expresión regular `r'.*Líneas Aéreas.*'` se utiliza para encontrar todas las líneas que contienen el texto "Líneas Aéreas".
     - `.*` antes y después de "Líneas Aéreas" significa que puede haber cualquier cantidad de caracteres (incluidos ninguno) antes o después de la frase clave.
     - Esto asegura que capturamos la línea completa que contiene la frase "Líneas Aéreas".
3. **Impresión de resultados**:
   - `print(a)` imprime la lista de líneas que han sido encontradas en el texto.

- Como resultado, se obtendrá una lista que incluye todas las líneas aéreas mencionadas en la variable `airlines`.

Agrupación

Con los paréntesis () se pueden agrupar varios caracteres, lo que resulta útil para:

- Aplicar un mismo cuantificador a todo el grupo.
- Capturar ese grupo, es decir, permitir su uso posterior (ya sea en la misma expresión regular de búsqueda o en el reemplazo).

```
# Ejemplo 1: Buscar dos ocurrencias consecutivas de "for"
a = re.search(r"(for ){2}", "Siempre busca for for el lado brillante de la vida")
print(a)

# Ejemplo 2: Buscar cualquier texto seguido de una repetición inmediata del mismo texto
a = re.search(r"(.+) \1", "Siempre busca for for el lado brillante de la vida")
print(a)

<re.Match object; span=(14, 22), match='for for '>
<re.Match object; span=(14, 21), match='for for'>
```

Explicación del Código

Este código demuestra cómo se pueden usar los paréntesis `()` en expresiones regulares para agrupar caracteres.

1. **Ejemplo 1: Buscar ocurrencias repetidas**:
   - `a = re.search(r"(for ){2}", "Siempre busca for for el lado brillante de la vida")`:
     - Se busca dos ocurrencias consecutivas de la palabra "for" en la cadena.
     - La expresión `(for ){2}` indica que se espera encontrar la palabra "for" seguida de un espacio exactamente dos veces.
2. **Ejemplo 2: Capturar repeticiones**:
   - `a = re.search(r"(.+) \1", "Siempre busca for for el lado brillante de la vida")`:
     - Aquí se busca cualquier texto (`(.+)`) seguido de un espacio y luego se repite exactamente el mismo texto (`\1`).
     - Esto captura patrones de texto que se repiten inmediatamente después de su primera aparición.

Ejercicios

Capturar por un lado el ancho y por otro el alto en estas medidas:

```python
# Capturar ancho y alto en medidas de formato "ancho x alto"
a = re.findall(r"(\d+)x(\d+)", "1280x720")
print(a)  # Salida: [('1280', '720')]

a = re.findall(r"(\d+)x(\d+)", "1920x1600")
print(a)  # Salida: [('1920', '1600')]

a = re.findall(r"(\d+)x(\d+)", "1024x768")
print(a)  # Salida: [('1024', '768')]
```

```
[('1280', '720')]
[('1920', '1600')]
[('1024', '768')]
```

Explicación del Código

Este código utiliza expresiones regulares para capturar el ancho y el alto en medidas dadas en formato "ancho x alto".

1. **Uso de `re.findall`**:
   - La función `re.findall` busca todas las coincidencias del patrón especificado en la cadena de texto.
   - Devuelve una lista de tuplas, donde cada tupla contiene los grupos capturados.
2. **Patrón de la expresión regular**:
   - `r"(\d+)x(\d+)"`:
     - `\d+` captura una o más cifras (números).
     - `()` se utilizan para agrupar las cifras, lo que permite capturar tanto el ancho como el alto por separado.
     - La letra `x` en medio actúa como un separador fijo entre el ancho y el alto.

Ahora, aparte de capturar los errores, queremos sustituirlos por las palabras correctas:

```python
# Texto con errores
texto = """todo estaba oscuro
era to más parecido
una de tas mejores obras
según decían, «tes asustaban»"""

# Realizamos las sustituciones para corregir los errores
resultados = re.sub(r'\bto\b', 'todo', texto)  # Sustituir "to" por "todo"
resultados = re.sub(r'\btas\b', 'las', resultados)  # Sustituir "tas" por "las"
resultados = re.sub(r'\btes\b', 'las', resultados)  # Sustituir "tes" por "las"

# Imprimir el texto corregido
print(resultados)
```

```
todo estaba oscuro
era todo más parecido
una de las mejores obras
según decían, «las asustaban»
```

Explicación del Código

Este código utiliza expresiones regulares para corregir palabras erróneas en un texto dado.

1. **Texto Original**:
   - Se define un bloque de texto que contiene varias oraciones con errores tipográficos o gramaticales.
2. **Uso de `re.sub`**:
   - La función `re.sub(pattern, repl, string)` busca todas las ocurrencias que coincidan con el `pattern` en el `string` y las reemplaza por `repl`.
3. **Sustituciones Específicas**:
   - `re.sub(r'\bto\b', 'todo', texto)`:
     - Busca la palabra `to` (con límites de palabra) y la reemplaza por `todo`.
   - `re.sub(r'\btas\b', 'las', resultados)`:
     - Busca la palabra `tas` y la reemplaza por `las`.
   - `re.sub(r'\btes\b', 'las', resultados)`:
     - Busca la palabra `tes` y la reemplaza por `las`.
   - Las expresiones regulares utilizan `\b` para asegurar que se está buscando una palabra completa y no una coincidencia dentro de otra palabra.

Grupos Anidados

Los grupos se pueden anidar en expresiones regulares, lo que significa que un fragmento de una cadena puede pertenecer a dos grupos diferentes. Esto es útil cuando queremos capturar información jerárquica o relacionada dentro de una misma expresión.

Imagina que en las siguientes cadenas queremos capturar tanto el año como el mes:

```python
# Texto de ejemplo con meses y años
texto = """Jan 1987
May 1969
Aug 2011"""

# Usamos una expresión regular para capturar el mes y el año
resultados = re.findall(r"(\w+) (\d{4})", texto)
print(resultados)
```

```
[('Jan', '1987'), ('May', '1969'), ('Aug', '2011')]
```

Explicación del Código

Este código utiliza expresiones regulares para capturar el mes y el año de un texto que contiene varias fechas.

1. **Texto de Ejemplo**:
   - Se define un bloque de texto que contiene varias fechas en formato "Mes Año", con cada par en una nueva línea.
2. **Uso de `re.findall`**:
   - La función `re.findall(pattern, string)` busca todas las coincidencias del patrón especificado en la cadena dada.
   - Devuelve una lista de todas las coincidencias encontradas.
3. **Expresión Regular**:
   - `r"(\w+) (\d{4})"`:
     - `(\w+)`: Captura el mes, que consiste en uno o más caracteres alfanuméricos (letras del mes).
     - `(\d{4})`: Captura el año, que consiste en exactamente cuatro dígitos.
     - El espacio entre ambos grupos permite que la expresión regular reconozca el patrón adecuado de "Mes Año".
4. **Salida**:
   - Al ejecutar el código, `print(resultados)` mostrará una lista de tuplas, donde cada tupla contiene el mes y el año.
   - Por ejemplo, para el texto dado, la salida será:

```
[('Jan', '1987'), ('May', '1969'), ('Aug', '2011')]
```