Status	Finished
Started	Thursday, 7 November 2024, 10:55 AM
Completed	Thursday, 7 November 2024, 11:01 AM
Duration	5 mins 38 secs
Marks	4.00/4.00
Grade	**10.00** out of 10.00 (**100**%)

Information

El módulo `re`

Python tiene un módulo llamado **`re`** para trabajar con expresiones regulares. Este módulo ofrece diferentes funciones para buscar un patrón expresado por una expresión regular en una string, e incluso para sustituir por otras substrings las apariciones que encajen con un patrón determinado.

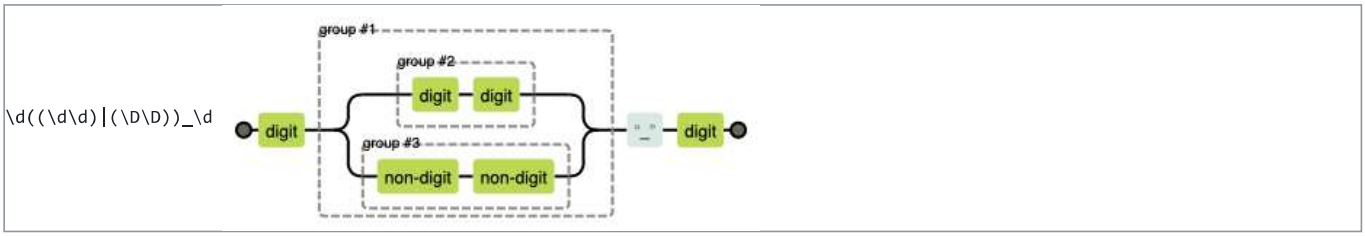
Muchas de estas funciones devuelven un objeto de coincidencia que contiene diversa información sobre la ocurrencia encontrada. Por ejemplo, la función `match()` busca la ocurrencia de un patrón al principio de una string y devuelve un *objeto* ***`match`***, si la hay, o None, si no la hay:

```
import re
pattern = '(\d\d)(-|/)(\d\d)(\\2)(\d{4})'
test_string = '30/06/1970 es una fecha muy popular'
result = re.match(pattern, test_string)
if result:
    print(result)          # <_sre.SRE_Match object; span=(0, 10), match='30/06/1970'>
    print(result.start())  # 0
    print(result.end())    # 10
    print(result.span())   # (0, 10)
    print(result.group(0)) # 30/06/1970
    print(result.group(5)) # 1970
```

Como vemos, los objetos de coincidencia ofrecen diferentes métodos para extraer la información de un objeto de ese tipo:

`start()`	Índice del comienzo de la ocurrencia encontrada, que para el método `match()` es siempre 0
`end()`	Índice de finalización de la ocurrencia, la cual va, como es usual, desde `start()` hasta `end()-1`, ambos inclusive
`span()`	Una tupla con los valores de `start()` y `end()`
`group()`	La substring que ha encajado con el grupo cuyo número se pasa por parámetro, además, `group(0)` devuelve la ocurrencia completa

Hay que tener cuidado con el uso del método `group()`; el grupo 0 siempre existe pero, dependiendo de la expresión, otros grupos podrían no existir (se retorna None), por lo que habría que preguntar si existen, antes de intentar acceder a ellos. En el siguiente ejemplo, los grupos 2 y 3 son excluyentes, de manera que si en una ocurrencia aparece uno de ellos, no puede aparecer el otro. Por ejemplo, dado el siguiente patrón:



El grupo 2 está en la string "123_4", donde no está el grupo 3, y el grupo 3 está en la string "1AB_4", donde no está el grupo 2.

Por el contrario, varias substrings pueden encajar con un mismo grupo si éste se encuentra dentro de una estructura repetitiva (p.e. con *): `group()` nos devolverá **solo la última** de tales substrings.

Information

La función `search()`

Si la función `match()` devuelve un objeto match para la ocurrencia del patrón al principio de la string de búsqueda, la función `search()` devuelve un objeto `match` para la primera ocurrencia del patrón en la string de búsqueda, independientemente de la posición donde se encuentre. Si no hay ninguna ocurrencia del patrón en la string de búsqueda, devuelve None, como la función `match()`:

```
import re
pattern = '\d\d(-|/)\d\d\\1\d{4}'
test_string = 'palabra1 30/06/1970 PALABRA2 12-12-2008 palabra3'
result = re.search(pattern, test_string)
print(result)
```

El ejemplo anterior muestra:

```
<_sre.SRE_Match object; span=(9, 19), match='30/06/1970'>
```

Information

La función `finditer()`

La función `finditer()` devuelve una secuencia iterable de objetos *match* con todas las ocurrencias (no solapadas) del patrón en la string donde se busca:

```
import re
```

```
pattern = '\d\d(-|/)\d\d\\1\d{4}'
test_string = 'palabra 1 30/06/1970 PALABRA2 12-12-2008 palabra3'
```

```
for match in re.finditer(pattern, test_string):
    print(match)
    print(match.start())
    print(match.end())
    print(match.span())
    print(match.group(0))
```

En el ejemplo anterior se muestra:

```
<re.Match object; span=(10, 20), match='30/06/1970'>
10
20
(10, 20)
30/06/1970
<re.Match object; span=(30, 40), match='12-12-2008'>
30
40
(30, 40)
12-12-2008
```

Question 1

Complete

Mark 1.00 out of 1.00

¿Qué función devuelve una secuencia de objetos de tipo *match*?

Select one:

- search()
- finditer()
- match()

Question 2

Complete

Mark 1.00 out of 1.00

¿Qué función busca la primera ocurrencia de una substring coincidente con el patrón?

Select one:

- match()
- finditer()
- search()

Information

La función `findall()`

La función `findall()` devuelve una lista con las ocurrencias en la string de búsqueda correspondientes a todos los grupos del patrón.

```
import re
```

```
pattern = '\d\d-\d\d-\d{4}'
test_string = 'palabra1 30-06-1970 PALABRA2 12-12-2008 palabra3'
```

```
for occurrence in re.findall(pattern, test_string):
    print(occurrence)
```

En este primer ejemplo no hemos introducido grupos en el patrón. El bucle itera, mostrando una ocurrencia cada vez, sobre la lista devuelta por la llamada a `findall()`, que es la siguiente.

```
['30-06-1970', '12-12-2008']
```

Hay que resaltar que la función `findall()` devuelve **los grupos** encontrados en cada ocurrencia. En el ejemplo anterior devuelve una lista de strings, cada una de las cuales es una ocurrencia completa correspondiente al grupo 0 (patrón completo). No obstante, si cambiamos el patrón por

```
'((\d\d)-(\d\d)-(\d{4}))'
```

lo que obtenemos es

```
[('30-06-1970', '30', '06', '1970'), ('12-12-2008', '12', '12', '2008')]
```

O sea, una lista de tuplas en la que cada una representa una ocurrencia y cada string corresponde a un grupo en esa ocurrencia. En el ejemplo, la primera string de cada tupla es la ocurrencia completa gracias a que hemos encerrado todo el patrón entre paréntesis, pues el grupo 0 solo se incluye (en lugar de la tupla) si, como en el primer ejemplo, no hay grupos definidos en el patrón. Por ello, si quitáramos los paréntesis más externos obtendríamos:

```
[('30', '06', '1970'), ('12', '12', '2008')]
```

Information

La función `split()`

La función `split()` divide la string de búsqueda en una lista de substrings usando las ocurrencias del patrón como separadores:

```
import re
pattern = '\d\d-\d\d-\d{4}'
test_string = 'palabra1 30-06-1970 PALABRA2 12-12-2008 palabra3'
print(re.split(pattern, test_string))
```

El ejemplo anterior muestra la lista:

```
['palabra1 ', ' PALABRA2 ', ' palabra3']
```

Como con `findall()`, hay que tener cuidado con la definición de grupos, ya que de haberlos se devuelven también. Si en el ejemplo anterior cambiamos el patrón por

```
'(\d\d)-\d\d-(\d{4})'
```

obtenemos

```
['palabra1 ', '30', '1970', ' PALABRA2 ', '12', '2008', ' palabra3']
```

Nótese que no aparece el mes de la fecha porque no se ha definido grupo para él.

Information

La función `sub()`

La función **`sub()`** sirve para sustituir las ocurrencias del patrón en la string de búsqueda por un nuevo valor:

```
import re
```

```
pattern = '(\d\d)-(\d\d)-(\d{4})'
test_string = 'palabra1 30-06-1970 PALABRA2 12-10-2008 palabra3'
new_string = re.sub(pattern, '<DATE>', test_string)
print(new_string)
```

El resultado del ejemplo anterior es:

```
"palabra1 <DATE> PALABRA2 <DATE> palabra3"
```

La string con el nuevo valor puede contener referencias a grupos de la ocurrencia que vamos a sustituir:

```
import re
```

```
pattern = '(\d\d)-(\d\d)-(\d{4})'
test_string = 'palabra1 30-06-1970 PALABRA2 12-10-2008 palabra3'
new_string = re.sub(pattern, '\\2/\\1/\\3', test_string)
print(new_string)
```

El resultado del ejemplo es:

```
"palabra1 06/30/1970 PALABRA2 10/12/2008 palabra3"
```

Con ello hemos conseguido intercambiar los grupos 1 y 2 de cada fecha, y sustituir los guiones por barras.

Existe una función parecida llamada **`subn()`** que, en lugar de devolver la string modificada, devuelve una tupla en la que el primer elemento es la string modificada y el segundo el número de ocurrencias que se han sustituido.

Question 3

Complete

Mark 1.00 out of 1.00

De las siguientes funciones, marque las que devuelven una lista de strings.

Select one or more:

- [ ] sub()
- [ ] finditer()
- [x] split()
- [x] findall()

Information

Flags

A todas las funciones de manejo de expresiones regulares se les pueden añadir como parámetros ciertos flags que modifican su comportamiento. Por ejemplo, el siguiente código:

```
import re
pattern = 'bra\d'
test_string = 'palabra1 30-06-1970 PALABRA2 12-10-2008 palabra3'
print(re.findall(pattern, test_string))
```

da como resultado:

```
['bra1', 'bra3']
```

Pero si añadimos un parámetro con el flag re.IGNORECASE:

```
import re
pattern = 'bra\d'
test_string = 'palabra1 30-06-1970 PALABRA2 12-10-2008 palabra3'
print(re.findall(pattern, test_string, re.IGNORECASE))
```

se obtiene:

```
['bra1', 'BRA2', 'bra3']
```

pues la búsqueda se realiza ignorando la diferencia entre mayúsculas y minúsculas.

Existen 6 posibles flags, cada uno con una versión larga y una abreviada, de una sola letra:

ASCII, A

Hace que las secuencias de escape \w, \b, \s y \d busquen sólo caracteres ASCII

DOTALL, S

Hace que el metacarácter '.' incluya los caracteres de salto de línea ('\n'). Por omisión el '.' significa "cualquier carácter que no sea un salto de línea".

IGNORECASE, I

Ignora la diferencia entre mayúsculas y minúsculas

LOCALE, L

Tiene en cuenta las convenciones locales del lenguaje.

MULTILINE, M

Búsqueda en strings multilínea. Hace que los metacaracteres ^ y $ coincidan, respectivamente, con el comienzo y fin de cada línea en una string multilínea. Por defecto, coinciden con el comienzo y fin de la string. Por ejemplo, el siguiente código:

```
import re
pattern = "^a\w+"
test_string = """amor
agua
casa
azúcar
"""
print(re.findall(pattern, test_string))
```

da como resultado ['amor'], pero si añadimos el flag multiline:

```
print(re.findall(pattern, test_string, re.M))
```

obtenemos: ['amor', 'agua', 'azúcar']

VERBOSE, X (por 'extended')

Facilita usar expresiones más legibles, permitiendo usar espacios para darles formato e introducir comentarios para explicarlas. Por ejemplo:

```
pattern = """
 &[#]               # Comienzo de un referencia numérica
 (
     0[0-7]+        # Formato octal
   | [0-9]+         # Formato decimal
   | x[0-9a-fA-F]+  # Formato Hexadecimal
 )
 ;                  # Punto y coma final
"""
```

en vez de:

```
pattern = "&[#](0[0-7]+|[0-9]+|x[0-9a-fA-F]+);"
```

Si se quieren usar varios flags, se pueden combinar con el operador '|':

```
print(re.findall(pattern, test_string, re.IGNORECASE | re.DOTALL | re.VERBOSE))
```

Question 4

Complete

Mark 1.00 out of 1.00

Empareje cada flag del módulo re de Python con su descripción.

DOTALL	Hace que el metacarácter '.' incluya los caracteres de salto de línea
LOCALE	Tiene en cuenta las especificidades del idioma local
VERBOSE	Facilita usar expresiones más legibles, permitiendo usar espacios para darles formato e introducir comentarios para explicarlas
MULTILINE	Hace que los metacarácteres ^ y $ coincidan, respectivamente, con el comienzo y fin de cada línea en una string multilínea
IGNORECASE	Se busca sin diferenciar entre mayúsculas y minúsculas
ASCII	Hace que las secuencias de escape \w, \b, \s y \d busquen solo carácteres ASCII