In [26]:
```
# importing re module
import re
```

This Python code uses regular expressions to search for the word “portal” in the given string and then prints the start and end indices of the matched word within the string.

In [25]:
```
import re

s = 'GeeksforGeeks: A computer science portal for geeks'

match = re.search(r'portal', s)

print('Start Index:', match.start())
print('End Index:', match.end())
```

```
Start Index: 34
End Index: 40
```

Note: Here r character (r’portal’) stands for raw, not regex. The raw string is slightly different from a regular string, it won’t interpret the \ character as an escape character. This is because the regular expression engine uses \ character for its own escaping purpose.

Before starting with the Python regex module let’s see how to actually write regex using metacharacters or special sequences.

Metacharacters

Metacharacters are the characters with special meaning.

To understand the RE analogy, Metacharacters are useful and important.

1. \ – Backslash The backslash () makes sure that the character is not treated in a special way. This can be considered a way of escaping metacharacters.

For example, if you want to search for the dot(.) in the string then you will find that dot(.) will be treated as a special character as is one of the metacharacters (as shown in the above table). So for this case, we will use the backslash() just before the dot(.) so that it will lose its specialty. See the below example for a better understanding.

Example: The first search (re.search(r'.', s)) matches any character, not just the period, while the second search (re.search(r'.', s)) specifically looks for and matches the period character.

In [30]:
```
import re

s = 'geeks.forgeeks'

# without using \
match = re.search(r'.', s)
print(match)

# using \
match = re.search(r'\.', s)
print(match)
```

```
<re.Match object; span=(0, 1), match='g'>
<re.Match object; span=(5, 6), match='.'>
```

1. [] – Square Brackets Square Brackets ([]) represent a character class consisting of a set of characters that we wish to match. For example, the character class [abc] will match any single a, b, or c.

We can also specify a range of characters using – inside the square brackets. For example,

[0, 3] is sample as [0123] [a-c] is same as [abc] We can also invert the character class using the caret(^) symbol. For example,

[^0-3] means any number except 0, 1, 2, or 3 [^a-c] means any character except a, b, or c Example: In this code, you're using regular expressions to find all the characters in the string that fall within the range of 'a' to 'm'. The re.findall() function returns a list of all such characters. In the given string, the characters that match this pattern are: 'c', 'k', 'b', 'f', 'j', 'e', 'h', 'l', 'd', 'g'.

In [31]:
```
import re

string = "The quick brown fox jumps over the lazy dog"
pattern = "[a-m]"
result = re.findall(pattern, string)

print(result)
```

```
['h', 'e', 'i', 'c', 'k', 'b', 'f', 'j', 'm', 'e', 'h', 'e', 'l', 'a', 'd', 'g']
```

1. ^ – Caret Caret (^) symbol matches the beginning of the string i.e. checks whether the string starts with the given character(s) or not. For example –

^g will check if the string starts with g such as geeks, globe, girl, g, etc. ^ge will check if the string starts with ge such as geeks, geeksforgeeks, etc. Example: This code uses regular expressions to check if a list of strings starts with "The". If a string begins with "The," it's marked as "Matched" otherwise, it's labeled as "Not matched".

In [32]:
```
import re
regex = r'^The'
strings = ['The quick brown fox', 'The lazy dog', 'A quick brown fox']
for string in strings:
        if re.match(regex, string):
                print(f'Matched: {string}')
        else:
                print(f'Not matched: {string}')
```

```
Matched: The quick brown fox
Matched: The lazy dog
Not matched: A quick brown fox
```

1. – $DollarDollar()$ symbol matches the end of the string i.e checks whether the string ends with the given character(s) or not. For example-

$swillcheckforthestringthatendswithasuchasgeeks, ends, s, etc. ks$ will check for the string that ends with ks such as geeks, geeksforgeeks, ks, etc. Example: This code uses a

regular expression to check if the string ends with “World!”. If a match is found, it prints “Match found!” otherwise, it prints “Match not found”.

In [36]:
```
import re

string = "Hello World"
pattern = r"World$"

match = re.search(pattern, string)
if match:
        print("Match found!")
else:
        print("Match not found.")
```

```
Match found!
```

1. . – Dot Dot(.) symbol matches only a single character except for the newline character (\n). For example –

a.b will check for the string that contains any character at the place of the dot such as acb, acbd, abbb, etc .. will check if the string contains at least 2 characters Example: This code uses a regular expression to search for the pattern “brown.fox” within the string. The dot (.) in the pattern represents any character. If a match is found, it prints “Match found!” otherwise, it prints “Match not found”.

In [39]:
```
import re

string = "The quick brown fox jumps over the lazy dog."
pattern = r"brown.fox"

match = re.search(pattern, string)
if match:
        print("Match found!")
else:
        print("Match not found.")
```

```
Match found!
```

1. | – Or Or symbol works as the or operator meaning it checks whether the pattern before or after the or symbol is present in the string or not. For example –

a|b will match any string that contains a or b such as acd, bcd, abcd, etc.

1. ? – Question Mark The question mark (?) is a quantifier in regular expressions that indicates that the preceding element should be matched zero or one time. It allows you to specify that the element is optional, meaning it may occur once or not at all. For example,

ab?c will be matched for the string ac, acb, dabc but will not be matched for abbc because there are two b. Similarly, it will not be matched for abdc because b is not followed by c.

8. – *Star Star* () symbol matches zero or more occurrences of the regex preceding the * symbol. For example –

ab*c will be matched for the string ac, abc, abbbc, dabc, etc. but will not be matched for abdc because b is not followed by c.

1. • – Plus Plus (+) symbol matches one or more occurrences of the regex preceding the + symbol. For example –

ab+c will be matched for the string abc, abbc, dabc, but will not be matched for ac, abdc, because there is no b in ac and b, is not followed by c in abdc.

1. {m, n} – Braces Braces match any repetitions preceding regex from m to n both inclusive. For example –

a{2, 4} will be matched for the string aaab, baaaac, gaad, but will not be matched for strings like abc, bc because there is only one a or no a in both the cases.

1. () – Group Group symbol is used to group sub-patterns. For example –

(a|b)cd will match for strings like acd, abcd, gacd, etc.

In [40]:
```
import re
string = """Hello my Number is 123456789 and
                        my friend's number is 987654321"""
regex = '\d+'

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

```
['123456789', '987654321']
```

Special Sequence	Description	Examples	
\A	Matches if the string begins with the given character	\Afor	for geeks
			for the world
\b	Matches if the word begins or ends with the given character. \b(string) will check for the beginning of the word and (string)\b will check for the ending of the word.	\bge	geeks
			get
\B	It is the opposite of the \b i.e. the string should not start or end with the given regex.	\Bge	together
			forge
\d	Matches any decimal digit, this is equivalent to the set class [0-9]	\d	123
			gee1

\D	Matches any non-digit character, this is equivalent to the set class [^0-9]	\D	geeks
			geek1
\s	Matches any whitespace character.	\s	gee ks
			a bc a
\S	Matches any non-whitespace character	\S	a bd
			abcd
\w	Matches any alphanumeric character, this is equivalent to the class [a-zA-Z0-9_].	\w	123
			geeKs4
\W	Matches any non-alphanumeric character.	\W	>$
			gee<>
\Z	Matches if the string ends with the given regex	ab\Z	abcdab
			abababab

Function	Description
re.findall()	finds and returns all matching occurrences in a list
re.compile()	Regular expressions are compiled into pattern objects
re.split()	Split string by the occurrences of a character or a pattern.
re.sub()	Replaces all occurrences of a character or patter with a replacement string.
re.escape()	Escapes special character
re.search()	Searches for first occurrence of character or pattern

In [41]:
```
import re
string = """Hello my Number is 123456789 and
                        my friend's number is 987654321"""
regex = '\d+'

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

```
['123456789', '987654321']
```

In [42]:
```
import re
p = re.compile('[a-e]')

print(p.findall("Aye, said Mr. Gibenson Stark"))
```

```
['e', 'a', 'd', 'b', 'e', 'a']
```

In [43]:
```
import re
p = re.compile('\d')
print(p.findall("I went to him at 11 A.M. on 4th July 1886"))

p = re.compile('\d+')
print(p.findall("I went to him at 11 A.M. on 4th July 1886"))
```

```
['1', '1', '4', '1', '8', '8', '6']
['11', '4', '1886']
```

In [45]:
```
import re

p = re.compile('\w')
print(p.findall("He said * in some_lang."))

p = re.compile('\w+')
print(p.findall("I went to him at 11 A.M., he \ said *** in some_language."))
```

```
p = re.compile('\W')
print(p.findall("he said *** in some_language."))
```

```
['H', 'e', 's', 'a', 'i', 'd', 'i', 'n', 's', 'o', 'm', 'e', '_', 'l', 'a', 'n',
'g']
['I', 'went', 'to', 'him', 'at', '11', 'A', 'M', 'he', 'said', 'in', 'some_languag
e']
[' ', ' ', '*', '*', '*', ' ', ' ', '.']
```

In [46]:
```
import re
p = re.compile('ab*')
print(p.findall("ababbaabbb"))
```

```
['ab', 'abb', 'a', 'abbb']
```

re.split(pattern, string, maxsplit=0, flags=0)

In [47]:
```
from re import split

print(split('\W+', 'Words, words , Words'))
print(split('\W+', "Word's words Words"))
print(split('\W+', 'On 12th Jan 2016, at 11:02 AM'))
print(split('\d+', 'On 12th Jan 2016, at 11:02 AM'))
```

```
['Words', 'words', 'Words']
['Word', 's', 'words', 'Words']
['On', '12th', 'Jan', '2016', 'at', '11', '02', 'AM']
['On ', 'th Jan ', ', at ', ':', ' AM']
```

In [48]:
```
import re
print(re.split('\d+', 'On 12th Jan 2016, at 11:02 AM', 1))
print(re.split('[a-f]+', 'Aey, Boy oh boy, come here', flags=re.IGNORECASE))
print(re.split('[a-f]+', 'Aey, Boy oh boy, come here'))
```

```
['On ', 'th Jan 2016, at 11:02 AM']
['', 'y, ', 'oy oh ', 'oy, ', 'om', ' h', 'r', '']
['A', 'y, Boy oh ', 'oy, ', 'om', ' h', 'r', '']
```

re.sub(pattern, repl, string, count=0, flags=0)

In [49]:
```
import re
print(re.sub('ub', '~*', 'Subject has Uber booked already',
                         flags=re.IGNORECASE))
print(re.sub('ub', '~*', 'Subject has Uber booked already'))
print(re.sub('ub', '~*', 'Subject has Uber booked already',
                         count=1, flags=re.IGNORECASE))
print(re.sub(r'\sAND\s', ' & ', 'Baked Beans And Spam',
                         flags=re.IGNORECASE))
```

```
S~*ject has ~*er booked already
S~*ject has Uber booked already
S~*ject has Uber booked already
Baked Beans & Spam
```

re.subn(pattern, repl, string, count=0, flags=0)

In [50]:
```
import re

print(re.subn('ub', '~*', 'Subject has Uber booked already'))

t = re.subn('ub', '~*', 'Subject has Uber booked already',
                         flags=re.IGNORECASE)
print(t)
```

```
print(len(t))
print(t[0])
```

```
('S~*ject has Uber booked already', 1)
('S~*ject has ~*er booked already', 2)
2
S~*ject has ~*er booked already
```

In [51]:
```
import re
print(re.escape("This is Awesome even 1 AM"))
print(re.escape("I Asked what is this [a-9], he said \t ^WoW"))
```

```
This\ is\ Awesome\ even\ 1\ AM
I\ Asked\ what\ is\ this\ \[a\-9\],\ he\ said\ \        \ \^WoW
```

In [52]:
```
import re
regex = r"([a-zA-Z]+) (\d+)"

match = re.search(regex, "I was born on June 24")
if match != None:
        print ("Match at index %s, %s" % (match.start(), match.end()))
        print ("Full match: %s" % (match.group(0)))
        print ("Month: %s" % (match.group(1)))
        print ("Day: %s" % (match.group(2)))

else:
        print ("The regex pattern does not match.")
```

```
Match at index 14, 21
Full match: June 24
Month: June
Day: 24
```

Set	Description
\{n,\}	Quantifies the preceding character or group and matches at least n occurrences.
*	Quantifies the preceding character or group and matches zero or more occurrences.
[0123]	Matches the specified digits (0, 1, 2, or 3)
[^arn]	matches for any character EXCEPT a, r, and n
\d	Matches any digit (0-9).
[0-5][0-9]	matches for any two-digit numbers from 00 and 59
\w	Matches any alphanumeric character (a-z, A-Z, 0-9, or _).
[a-n]	Matches any lower case alphabet between a and n.
\D	Matches any non-digit character.

[arn]	matches where one of the specified characters (a, r, or n) are present
[a-zA-Z]	matches any character between a and z, lower case OR upper case
[0-9]	matches any digit between 0 and 9

In [ ]: