What are regular expressions?

The Regex or Regular Expression is a way to define a pattern for searching or manipulating strings. We can use a regular expression to match, search, replace, and manipulate inside textual data.

Also, Regular expressions are instrumental in extracting information from text such as log files, spreadsheets, or even textual documents.

Example 1: Write a regular expression to search digit inside a string

In [1]:
```
import re
string = "My roll no. is 25"
a = re.findall(r"\d",string)
a
```

Out[1]:
```
['2', '5']
```

Understand this example

- We imported the RE module into our program
- Next, We created a regex pattern d to match any digit between 0 to 9.
- After that, we used the re.findall() method to match our pattern.
- In the end, we got two digits 2 and 5.

Use raw string to define a regex

Note: I have used a raw string to define a pattern like this r"d". Always write your regex as a raw string.

- As you may already know, the backslash has a special meaning in some cases because it may indicate an escape character or escape sequence. To avoid that always use a raw string.

Python regex methods

1. Compile Regex Pattern using re.compile()

We can compile a regular expression into a regex object to look for occurrences of the same pattern inside various target strings without rewriting it.

How to compile regex pattern

1. Write regex pattern in string format
2. Write regex pattern using a raw string. For example, a pattern to match any digit.

```
str_pattern = r'\d'
```

3. Pass a pattern to the compile() method

```
pattern = re.compile(r'\d{3})
```

4. It compiles a regular expression pattern provided as a string into a regex pattern object.
5. Use Pattern object to match a regex pattern
6. Use Pattern object returned by the compile() method to match a regex pattern.

```
res = pattern.findall(target_string)
```

Example to compile a regular expression

Now, let's see how to use the re.compile() with the help of a simple example.

- Pattern to compile: r'\d{3}'

What does this pattern mean?

- First of all, I used a raw string to specify the regular expression pattern.
- Next, \d is a special sequence and it will match any digit from 0 to 9 in a target string.

- Then the 3 inside curly braces mean the digit has to occur exactly three times in a row inside the target string.
- In simple words, it means to match any three consecutive digits inside the target string such as 236 or 452, or 782.

In [2]:
```
# target string
str1 = " Deepali lucky numbers are 894 234 456 829"

#pattern to find 3 consecutive digits
str_pattern = r"\d{3}"

#compile str_pattern to re.pattern object
regex_pattern = re.compile(str_pattern)

#type of compile
print(type(regex_pattern))

# find all the matches in the string 1
result = regex_pattern.findall(str1)
print(result)

# target string 2
str2 = " Harsh lucky numbers are 678 645 234 097"

# find all the matches in second string by reusing the same pattern
res2 = regex_pattern.findall(str2)
print(res2)

```

```
<class 're.Pattern'>
['894', '234', '456', '829']
['678', '645', '234', '097']
```

As you can see, we found four matches of “three consecutive” digits inside the first string.

Note:

- The re.compile() method changed the string pattern into a re.Pattern object that we can work upon.
- Next, we used the re.Pattern object inside a re.findall() method to obtain all the possible matches of any three consecutive digits inside the target string.
- Now, the same reagex_pattern object can be used similarly for searching for three consecutive digits in other target strings as well.

Why and when to use re.compile()

- Performance improvement
- Compiling regular expression objects is useful and efficient when the expression will be used several times in a single program.

Keep in mind that the compile() method is useful for defining and creating regular expressions object initially and then using that object we can look for occurrences of the same pattern inside various target strings without rewriting it which saves time and improves performance.

Readability

Another benefit is readability. Using re.compile() you can separate the definition of the regex from its use.

In [3]:
```
pattern = re.compile("str_pattern")
result = pattern.match(string)
```

is equivalent to

In [4]:
```
result = re.match("str_pattern", string)

```

so, you should use the compile() method when you're going to perform a lot of matches using the same pattern. Also, when you are searching for the same pattern over and over again and in multiple target strings

2. Regex Search using re.search()

- Python regex re.search() method looks for occurrences of the regex pattern inside the entire target string and returns the corresponding Match Object instance where the match found.
- The re.search() returns only the first match to the pattern from the target string. Use a re.search() to search pattern anywhere in the string.

Regex search example – look for a word inside the target string

Now, let's see how to use re.search() to search for the following pattern inside a string.

Pattern: \w{8}

What does this pattern mean?

- The \w is a regex special sequence that represents any alphanumeric character such as letters uppercase or lowercase, digits as well as the underscore character.
- Then the 8 inside curly braces mean the characters have to occur exactly 8 times in a row inside the target string

In simple words, it means to search any eight-letter word

In [5]:
```
str1 = "Deepali is a baseball player who was born on June 17"

# search() for eight-letter word
result = re.search(r"\w{8}",str1)

# print match object
print("Match object",result)

## print the matching word using group() method
print("Matching word: ", result.group())

```

```
Match object <re.Match object; span=(13, 21), match='baseball'>
Matching word:  baseball
```

Let's understand the above example.

- First of all, I used a raw string to specify the regular expression pattern. As you may already know, the backslash has a special meaning in some cases because it may indicate an escape character or escape sequence. To avoid that we used raw string.
- Also, we are not defining and compiling this pattern beforehand (like the compile method), The practice is to write the actual pattern in the string format.
- Next, we wrote a regex pattern to search for any eight-letter word inside the target string.
- Next, we passed this pattern to re.search() method to looks for occurrences of the pattern and it returned the re.Match object.
- Next, we used the group() method of a re.Match object to retrieve the exact match value i.e., baseball.

Regex search example find exact substring or word

In this example, we will find substring “ball” and “player” inside a target string.

In [6]:
```
str1 = "Deepali is a baseball player who was born on June 17"

# finding substring ball
result = re.search(r"ball",str1)

# match the substring
print(result.group())

# find exact word/substring surrounded by word boundary
result = re.search(r"\bball\b",str1)

if result:
    print(result)
# output None

# find word 'player'
result = re.search(r"\bplayer\b", str1)
print(result.group())
```

```
ball
player
```

When to use re.search()

The search() method will always match and return only the first occurrence of the pattern from the target string.

- Use it when you want to find the first match. The search method is useful for a quick match. I.e., As soon as it gets the first match, it will stop its execution. You will get performance benefits.
- Also, please use it when you want to check the possibility of a pattern in a long target string.
- Avoid using the search() method in the following cases
- To search all occurrence to a regular expression, please use the findall() method instead.
- To search at the start of the string, Please use the match() method instead. Also, read regex search() vs. match()
- If you want to perform search and replace operation in Python using regex, please use the re.sub() method.

Regex search groups or multiple patterns

- to search for multiple distinct patterns inside the same target string. Let's assume, we want to search the following two distinct patterns inside the target string at the same time.
- A ten-letter word
- Two consecutive digits To achieve this, Let's write two regular expression patterns.

Regex Pattern 1: \w{10}

It will search for any six-letter word inside the target string

Regex Pattern 2: \d{2}

Now each pattern will represent one group. Let's add each group inside a parenthesis ( ). In our case r"(\w{10}).+(\d{2})"

On a successful search, we can use match.group(1) to get the match value of a first group and match.group(2) to get the match value of a second group.

In [7]:
```
str1 = "Deepali is a basketball player who was born on June 17"

# two group enclosed in seprate (and) brackets
result = re.search(r"(\w{10}).+(\d{2})",str1)

# Extract the matches using group()

# print ten-letter word
print(result.group(1))

# # print two digit number
print(result.group(2))


```

```
basketball
17
```

Let's understand this example

- We enclosed each pattern in the separate opening and closing bracket.

- I have added the .+ metacharacter before the second pattern. the dot represents any character except a new line and the plus sign means that the preceding pattern is repeating one or more times. So .+ means that before the first group, we have a bunch of characters that we can ignore
- Next, we used the group() method to extract two matching values.
- **Note:** The group() method returned two matching values because we used two patterns.

Search multiple words using regex

Let's take another example and search any three words surrounded by space using regex. Let's search words "Deepali", "player", "born" in the target string.

- Use | (pipe) operator to specify multiple patterns.

In [8]:
```
str1 = "Deepali is a baseball player who was born on June 17, 1993."

# search() for eight-letter word surrounded by space
# \b is used to specify word boundary
result = re.findall(r"\bDeepali\b|\bplayer\b|\bborn\b",str1)

print(result)
```

```
['Deepali', 'player', 'born']
```

Case insensitive regex search

There is a possibility that the string contains lowercase and upper case words or words with a combination of lower case and uppercase letters.

For example, you want to search a word using regex in a target string, but you don't know whether that word is in uppercase or lowercase letter or a combination of both. Here you can use the re.IGNORECASE flag inside the search() method to perform case-insensitive searching of a regex pattern.

In [9]:
```
str1 = "Deepali is a Baseball player who was born on June 17, 1993."

# case sesitive searching
result = re.search(r"deepali",str1)
print("matching word :",result)

# print case-sesitive using ignorecase
result = re.search(r"deepali",str1,re.IGNORECASE)
print("matching word: ",result.group())
```

```
matching word : None
matching word:  Deepali
```

3. Re.Match():

Match regex pattern at the beginning of the string

Now, Let's see the example to match any four-letter word at the beginning of the string. (Check if the string starts with a given pattern).

Pattern to match: \w{4}

What does this pattern mean?

- The \w is a regex special sequence that represents any alphanumeric character meaning letters (uppercase or lowercase), digits, and the underscore character.
- Then the 4 inside curly braces say that the character has to occur exactly four times in a row (four consecutive characters).
- In simple words, it means to match any four-letter word at the beginning of the following string.

In [10]:
```
target_string = "Sila is a basketball player who was born on June 17, 1993"
result = re.match(r"\w{4}",target_string)

# printing the match object
print("Matching obj",result)

# extract match value:
print("match value:",result.group())

```

```
Matching obj <re.Match object; span=(0, 4), match='Sila'>
match value: Sila
```

Match regex pattern anywhere in the string

Let's assume you want to match any six-letter word inside the following target string

- target_string = "Jessa loves Python and pandas"

If you use a match() method to match any six-letter word inside the string you will get None because it returns a match only if the pattern is located at the beginning of the string. And as we can see the six-letter word is not present at the start.

So to match the regex pattern anywhere in the string you need to use either search() or findall() method of a RE module.

In [11]:
```
target_string = "dea loves Python and pandas"

pattern = r"\w{6}"

#match method()
result = re.match(pattern,target_string)
print(result)

# search()method
result = re.search(pattern,target_string)
print(result.group())

# findall()method
result = re.findall(pattern,target_string)
print(result)
```

```
None
Python
['Python', 'pandas']
```

Match regex at the end of the string

Sometimes we want to match the pattern at the end of the string. For example, you want to check whether a string is ending with a specific word, number or, character.

Using a dollar ($) metacharacter we can match the regular expression pattern at the end of the string.

Example to match the four-digit number at the end of the string

In [12]:
```
target_string = "Emma is a basketball player who was born on June 17, 1993"

# Match at the end
result = re.search(r"\d{4}$",target_string)
print(result)

print("Match num:",result.group())
```

```
<re.Match object; span=(53, 57), match='1993'>
Match num: 1993
```

Match the exact word or string

In [13]:
```
#to match the word “player” in the target string.
target_string = "Emma is a basketball player who was born on June 17, 1993"

result = re.findall(r"player",target_string)
print(result)
```

```
['player']
```

Understand the Match object

the match() and search() method returns a re.Match object if a match found. Let’s see the structure of a re.Match object.

```
re.Match object; span=(0, 4), match='Emma'

```

This re.Match object contains the following items.

- A span attribute that shows the locations at which the match starts and ends. i.e., is the tuple object contains the start and end index of a successful match.
- Save this tuple and use it whenever you want to retrieve a matching string from the target string
- Second, A match attribute contains an actual match value that we can retrieve using a group() method.

In [14]:
```
string = "Deepa and shila"

## Match five-letter word
result = re.match(r"\b\w{5}\b",string)

# printing entire match object
print(result)

# Extract Matching value
print(result.group())

# Start index of a match
print("start index is :",result.start())

# End index of a match
print("End index:",result.end())

# Start and end index of a match
position = result.span()
print(position)

# Use span to retrieve the matching string
print(string[position[0]:position[1]])
```

```
<re.Match object; span=(0, 5), match='Deepa'>
Deepa
start index is : 0
End index: 5
(0, 5)
Deepa
```

Match regex pattern that starts and ends with the given text

Let's assume you want to check if a given string starts and ends with a particular text. We can do this using the following two regex metacharacter with re.match() method.

- Use the caret metacharacter to match at the start
- Use dollar metacharacter to match at the end

- Now, let's check if the given string starts with the letter 'p' and ends with the letter 't'

In [15]:
```
import re

# string starts with letter 'p' ends with letter 's'
def starts_ends_with(str1):
    res = re.match(r'^(P).*(s)$', str1)
    if res:
        print(res.group())
    else:
        print('None')

str1 = "Pooja is for Python developers"
starts_ends_with(str1)
# Output 'PYnative is for Python developers'

str2 = "Pooja is for Python"
starts_ends_with(str2)
# Output None
```

```
Pooja is for Python developers
None
```

some common regex matching operations such as

- Match any character
- Match number
- Match digits
- match special characters

In [16]:
```
string = "Deepali 21 25"

# match any character
print(re.match(r'.',string))

# match all digit
print(re.findall(r"\d",string))

# match all numbers
# + indicate 1 or more occurence of \d
print(re.findall(r"\d+",string))

# match all special characters and symbol

str2 = "Deep@li $$$$@%^"
print(re.findall(r"\W",str2))
```

```
<re.Match object; span=(0, 1), match='D'>
['2', '1', '2', '5']
['21', '25']
['@', ' ', '$', '$', '$', '$', '@', '%', '^']
```

4.Regex Find All matches using findall() and finditer()

find all matches to a regex pattern

In this example, we will find all numbers present inside the target string. To achieve this, let's write a regex pattern.

Pattern: \d+

What does this pattern mean?

- The \d is a special regex sequence that matches any digit from 0 to 9 in a target string.
- The + metacharacter indicates number can contain at minimum one or maximum any number of digits.

In simple words, it means to match any number inside the following target string.

- target_string = "Emma is a basketball player who was born on June 17, 1993. She played 112 matches with scoring average 26.12 points per game. Her weight is 51 kg."

As we can see in the above string ’17’, ‘1993’, ‘112’, ’26’, ’12’, ’51’ number are present, so we should get all those numbers in the output.

In [17]:
```
target_string = "Emma is a basketball player who was born on June 17, 1993. She played 112 matches with scoring aver
result = re.findall(r"\d+",target_string)

# printing all the matches
print(result)
```

```
['17', '1993', '112', '26', '12', '51']
```

Note: First of all, I used a raw string to specify the regular expression pattern i.e r"\d+". As you may already know, the backslash has a special meaning in some cases because it may indicate an escape character or escape sequence to avoid that we must use raw string.

finditer method

The re.finditer() works exactly the same as the re.findall() method except it returns an iterator yielding match objects matching the regex pattern in a string instead of a list. It scans the string from left-to-right, and matches are returned in the iterator form. Later, we can use this iterator object to extract all matches.

In simple words, finditer() returns an iterator over MatchObject objects.

But why use finditer()?

In some scenarios, the number of matches is high, and you could risk filling up your memory by loading them all using findall(). Instead of that using the finditer(), you can get all possible matches in the form of an iterator object, which will improve performance.

It means, finditer() returns a callable object which will load results in memory when called.

example to find all two consecutive digits inside the target string.

In [18]:
```
target_string = "Emma is a basketball player who was born on June 17, 1993. She played 112 matches with scoring aver

#finditer() with regex pattern and target string
# \d{2} to match two consecutive digits
result = re.finditer(r"\d{2}",target_string)

# printing all match object

for match_obj in result:
    # print re.match obj
    print(match_obj)

    # exctract each mathcing number
    print(match_obj.group())
```

```
<re.Match object; span=(49, 51), match='17'>
17
<re.Match object; span=(53, 55), match='19'>
19
<re.Match object; span=(55, 57), match='93'>
93
<re.Match object; span=(70, 72), match='11'>
11
<re.Match object; span=(103, 105), match='26'>
26
<re.Match object; span=(106, 108), match='12'>
12
<re.Match object; span=(140, 142), match='51'>
51
```

Regex find all word starting with specific letters

In this example, we will see solve following 2 scenarios

- find all words that start with a specific letter/character
- find all words that start with a specific substring

Now, let's assume you have the following string:

> *target_string = "Jessa is a Python developer. She also gives Python programming training" Now let's find all word that starts with letter p. Also, find all words that start with substring 'py'

- Pattern: \b[p]\w+\b
- The \b is a word boundary, then p in square bracket [] means the word must start with the letter 'p'.
- Next, \w+ means one or more alphanumerical characters after a letter 'p'
- In the end, we used \b to indicate word boundary i.e. end of the word.

In [19]:
```
target_string = "Deepa is a Python developer. She also gives Python programming training"

# all words start with p
result1 = re.findall(r"\b[p]\w+\b",target_string,re.I)
print(result1)

# all words that start with substring "py"
result2 = re.findall(r"\bpy\w+\b",target_string,re.I)
print(result2)
```

```
['Python', 'Python', 'programming']
['Python', 'Python']
```

Regex to find all word that starts and ends with a specific letter

In this example, we will see solve following 2 scenarios

- find all words that start and ends with a specific letter
- find all words that start and ends with a specific substring

In [20]:
```
target_string = "Jessa is a Python developer. She also gives Python programming training"

# all word starts with letter 'p' and ends with letter 'g'
res1 = re.findall(r"\b[p]\w+[g]\b",target_string,re.I)
print(res1)

#  all words that start and ends with a specific substring
res2 = re.findall(r"\b[pt]\w+[g]\b",target_string,re.I)
print(res2)


target_string2 = "Deepali loves mango and orange"
## all word starts with substring 'ma' and ends with substring 'go'
res3 = re.findall(r"\bma\w+[go]\b",target_string2,re.I)

target_string3 = "Kelly loves banana and apple"
# all word starts or ends with letter 'a
res4 = re.findall(r"\b[a]\w+\b|\w+[a]\b",target_string3,re.I)
print(res4)
```

```
['programming']
['programming', 'training']
['banana', 'and', 'apple']
```

Regex to find all words containing a certain letter

In this example, we will see how to find words that contain the letter ‘i’.

In [21]:
```
target_string = "Jessa is a knows testing and machine learning"

# find all word that contain letter 'i'
print("1. Containing i letter words :",re.findall(r"\b\w*[i]\w*\b",target_string,re.I))

## find all word which contain substring 'ing'
print("2.Containing substring of ing :",re.findall(r"\b\w+[ing]\b",target_string,re.I))
```

```
1. Containing i letter words : ['is', 'testing', 'machine', 'learning']
2.Containing substring of ing : ['testing', 'learning']
```

Regex findall repeated characters

For example, you have a string: ""Jessa Erriika""

In [22]:
```
import re

target_string = "Jessa Erriika"
# This '\w' matches any single character
# and then its repetitions (\1*) if any.
matcher = re.compile(r"(\w)\1*")

for match in matcher.finditer(target_string):
    print(match.group(), end=", ")
# output J, e, ss, a, E, rr, ii, k, a,
```

```
J, e, ss, a, E, rr, ii, k, a,
```

5. re.split()

The Pythons re module's re.split() method split the string by the occurrences of the regex pattern, returning a list containing the resulting substrings.

Regex example to split a string into words

Now, let's see how to use re.split() with the help of a simple example. In this example, we will split the target string at each white-space character using the \s special sequence.

Let's add the + metacharacter at the end of \s. Now, The \s+ regex pattern will split the target string on the occurrence of one or more whitespace characters.

In [23]:
```
import re

target_string = "My name is maximums and my luck numbers are 12 45 78"
# split on white-space
word_list = re.split(r"\s+", target_string)
print(word_list)

```

```
['My', 'name', 'is', 'maximums', 'and', 'my', 'luck', 'numbers', 'are', '12', '45', '78']
```

Limit the number of splits

The maxsplit parameter of re.split() is used to define how many splits you want to perform.

In simple words, if the maxsplit is 2, then two splits will be done, and the remainder of the string is returned as the final element of the list.

So let's take a simple example to split a string on the occurrence of any non-digit. Here we will use the \D special sequence that matches any non-digit character.

In [24]:
```
target_string = "12-45-78"

# Split only on the first occurrence
# maxsplit is 1
result = re.split(r"\D",target_string,maxsplit=1)
print(result)

# Split only on the 3 occurrence
# maxsplit is 3
result2 = re.split(r"\D",target_string,maxsplit=3)
print(result2)

```

```
['12', '45-78']
['12', '45', '78']
```

Split string by multiple delimiters using regex

With the regex split() method, you will get more flexibility. You can specify a pattern for the delimiter, while with the string's split() method, you could have used only a fixed character or set of characters to split a string.

For example, using the regular expression re.split() method, we can split the string either by the comma or by space. Also, you can split a string based on multiple patterns.

Let's take a simple example to split the string either by the hyphen or by the comma.

Example to split string by two delimiters

In [25]:
```
1
2 target_string = "12,45,78,85-17-89"
3 # 2 delimiter - and ,
4 # use OR (|) operator to combine two pattern
5 result = re.split(r"-|,", target_string)
6 print(result)
7 # Output ['12', '45', '78', '85', '17', '89']
```

```
['12', '45', '78', '85', '17', '89']
```

Example to split string with 5 delimiters

Including the dot, comma, semicolon, a hyphen, and space followed by any amount of extra whitespace).

In [26]:
```
1 target_string = "Python   dot.com; is for, Python-developer"
2 # Pattern to split: [-;,.\s]\s*
3 result = re.split(r"[-;,.\s]\s*", target_string)
4 print(result)
```

```
['Python', 'dot', 'com', 'is', 'for', 'Python', 'developer']
```

Note: we used [ ] meta character to indicate a set of delimiter characters. The [] matches any single character in brackets. For example, [-;,.\s] will match either hyphen, comma, semicolon, dot, and a space character.

Split Strings into words with multiple word boundary delimiters

In this example, we will use the[\b\W\b]+ regex pattern to cater to any Non-alphanumeric delimiters. Using this pattern we can split string by multiple word boundary delimiters that will result in a list of alphanumeric/word tokens.

Note: The \W is a regex special sequence that matches any Non-alphanumeric character. Non-alphanumeric means no letter, digit, and underscore.

In [27]:
```
target_string = "PYnative! dot.com; is for, Python-developer?"
result = re.split(r"[\b\W\b]+", target_string)
print(result)

```

```
['PYnative', 'dot', 'com', 'is', 'for', 'Python', 'developer', '']
```

Split strings by delimiters and specific word

In [28]:
```
text = "12, and45,78and85-17and89-97"

# split by word and,space comma
print(re.split(r"and|[\s,-]+",text))
```

```
['12', '', '45', '78', '85', '17', '89', '97']
```

Regex split a string and keep the separators

In [29]:
```
target_string = "12-45-78."

# split on non-digit
print(re.split(r"\D+",target_string))

#Split on non-digit and keep the separators
# pattern written in parenthese
print(re.split(r"(\D+)",target_string))
```

```
['12', '45', '78', '']
['12', '-', '45', '-', '78', '.', '']
```

regex split string by ignoring case

There is a possibility that the string contains lowercase and upper case letters.

For example, you want to split a string on the specific characters or range of characters, but you don't know whether that character/word is an uppercase or lowercase letter or a combination of both. Here you can use the re.IGNORECASE or re.I flag inside the re.split() method to perform case-insensitive splits.

In [30]:
```
# without ignoring case
print(re.split("[a-z]+","7J8e7Ss3a"))

# With ignoring case
print(re.split("[a-z]+","7J8e7Ss3a",flags=re.IGNORECASE))

# Without ignoring case
print(re.split(r"emma", "Emma knows Python.EMMA loves Data Science"))

# With ignoring case
print(re.split(r"emma", "Emma knows Python.EMMA loves Data Science", flags=re.IGNORECASE))
```

```
['7J8', '7S', '3', '']
['7', '8', '7', '3', '']
['Emma knows Python.EMMA loves Data Science']
['', ' knows Python.', ' loves Data Science']
```

Split string by upper case words

For example, you have a string like "DEEPA loves PYTHON and ML", and you wanted to split it by uppercase words to get results like ['HELLO there', 'HOW are', 'YOU']

In [31]:
```
print(re.split(r"\s(?=[A-Z])", "DEEPA loves PYTHON and ML"))

```

```
['DEEPA loves', 'PYTHON and', 'ML']
```

6. Replace pattern using re.sub

Regex example to replace all whitespace with an underscore

Now, let's see how to use re.sub() with the help of a simple example. Here, we will perform two replacement operations

- Replace all the whitespace with a hyphen
- Remove all whitespaces

Let's see the first scenario first.

Pattern to replace: \s

In this example, we will use the \s regex special sequence that matches any whitespace character, short for [\t\n\x0b\r\f]

Let's assume you have the following string and you wanted to replace all the whitespace with an underscore.

In [32]:
```
target_str = "Deepa knows testing and machine learning"
result_str = re.sub(r"\s","_",target_str)

# string after replacement
print(result_str)
```

```
Deepa_knows_testing_and_machine_learning
```

Regex to remove whitespaces from a string

Now, let's move to the second scenario, where you can remove all whitespace from a string using regex. This regex remove operation includes the following four cases.

- Remove all spaces, including single or multiple spaces (pattern to remove \s+)
- Remove leading spaces (pattern to remove ^\s+)
- Remove trailing spaces (pattern to remove \s+$)
- Remove both leading and trailing spaces. (pattern to remove ^\s+|\s+$)

Example 1: Remove all spaces

In [33]:
```
target_str = "Deepa knows testing and machine learning \t."

# \s+ to remove all spaces
# + indicate 1 or more occurance of white space
print(re.sub(r"\s+","",target_str))
```

```
Deepaknowstestingandmachinelearning.
```

Example 2: Remove leading spaces

In [34]:
```
target_str = "Deepa knows testing and machine learning ."

# ^\s+ remove only leading spaces
# caret(^) matches only start of the string
print(re.sub(r"^\s+","",target_str))
```

```
Deepa knows testing and machine learning .
```

Example 3: Remove trailing spaces

In [35]:
```
target_str = "     Deepa knows testing and machine learning   \t\n"

## ^\s+$ remove only trailing spaces
# dollar ($) matches spaces only at the end of the string
print(re.sub(r"\s+$","",target_str))
```

```
     Deepa knows testing and machine learning
```

Example 4: Remove both leading and trailing spaces

In [36]:
```
target_str = "Deepa knows testing and machine learning ."

# ^\s+$ remove only trailing spaces
# \s+ to remove all spaces
# | operator combine both patterns
print(re.sub(r"^\s+|\s+$","",target_str))
```

```
Deepa knows testing and machine learning .
```

In [37]:
```

target_str = "Jessa Knows Testing    And Machine     Learning \t \n"

# \s+ to match all whitespaces
# replace them using single space " "
res_str = re.sub(r"\s+", " ", target_str)

# string after replacement
print(res_str)
# Output 'Jessa Knows Testing And Machine Learning'
```

```
Jessa Knows Testing And Machine Learning
```

Regex replacement function

For example, you want to replace all uppercase letters with a lowercase letter. To achieve this we need the following two things

A regular expression pattern that matches all uppercase letters and the replacement function will convert matched uppercase letters to lowercase.

Pattern to replace: [A-Z]

This pattern will match any uppercase letters inside a target string.

replacement function

You can pass a function to re.sub. When you execute re.sub() your function will receive a match object as the argument. If can perform replacement operation by extracting matched value from a match object.

If a replacement is a function, it is called for every non-overlapping occurrence of pattern. The function takes a single match object argument and returns the replacement string

So in our case, we will do the followings

- First, we need to create a function to replace uppercase letters with a lowercase letter
- Next, we need to pass this function as the replacement argument to the re.sub()
- Whenever re.sub() matches the pattern, It will send the corresponding match object to the replacement function
- Inside a replacement function, we will use the group() method to extract an uppercase letter and convert it into a lowercase letter

In [38]:
```
# replacement function to convert uppercase letter to lowercase

def convert_to_lower(match_obj):
    if match_obj.group() is not None:
        return match_obj.group().lower()

# Original String
str1 = "Deepa LOves PINEAPPLE DEssert and COCONUT Ice Cream"

# passing replacment function to re.sub()
print(re.sub(r"[A-Z]",convert_to_lower,str1))

```

```
deepa loves pineapple dessert and coconut ice cream
```

Regex replace group/multiple regex patterns

To understand this take the example of the following string

student_names = "Emma-Kelly Jessa Joy Scott-Joe Jerry"

Here, we want to find and replace two distinct patterns at the same time.

We want to replace each whitespace and hyphen(-) with a comma (,) inside the target string. To achieve this, we must first write two regular expression patterns.

- Pattern 1: \s matches all whitespaces
- Pattern 2: - matches hyphen(-)

In [39]:

```
1
2 # Original string
3 student_names = "Emma-Kelly Jessa Joy Scott-Joe Jerry"
4
5 # replace two pattern at the same time
6 # use OR (|) to separate two pattern
7 res = re.sub(r"(\s)|(-)", ",", student_names)
8 print(res)
9 # Output 'Emma,Kelly,Jessa,Joy,Scott,Joe,Jerry'
```

```
Emma,Kelly,Jessa,Joy,Scott,Joe,Jerry
```