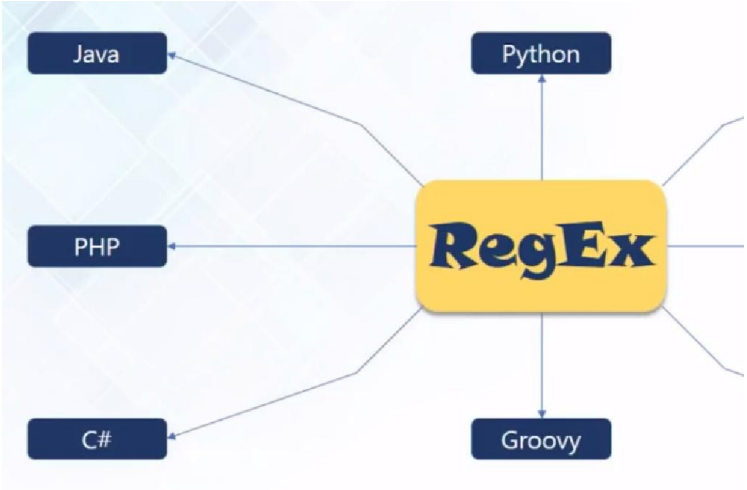
# Regular Expressions

## Why we use regular expressions?

We can use regular expression to solve problems in various areas. Consider the following situations

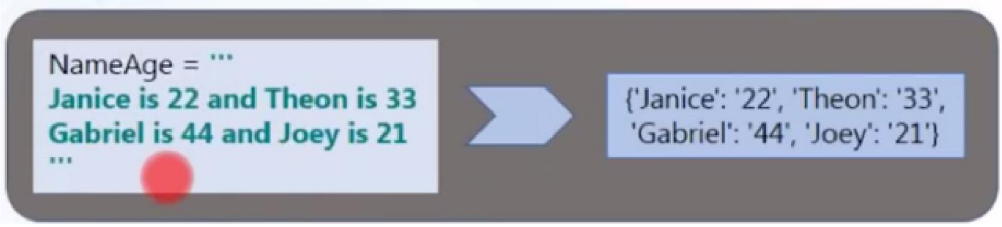
- We have a log file and we have to find out date & time from that log file, as we know format of log file is not understandable to every one, but we use regular expression and identify the pattern and find only date & time from log file.
- You have lots of email address and many of them might be fake, the valid email address having particular pattern or format, with the help of regular expression we can find out which email address are correct and which are fake email addresses.
- You have lots of phone numbers and many of them might not be correct, the valid phone numbers having specific pattern according to country, therefore using regular expression we can identify which one is correct and which one is fake.
- We have a student database that contain a name, age, address and area code, now we have lots of students from the particular area code say 59006, suppose these area code is updated from 59006 to 59076 then if do ie manually for every student then it takes lots of time, instead of that what we can do we can use regular expression and we can find out area code from student database and replace it with updated area code. So with regular expression we can replace particular string also.
- Regular expressions are compatible with many of the programming language like Java, PHP, Ruby, Python, Javascript etc.



## What are the Regular Expressions?

A regular expression is a **special text string for describing a search pattern.** So you can use regular expression for searching a treat string in a large amount of data, even you can verify the string is in given format or not, even you can format the data in proper form.

Suppose we have a large string that contain name and age




What are useful data that I can find out here is only name and age, rest of things I don't want and convert it to in a dictionary, for that we can identify the pattern in regular expression that will do the same. What should be the pattern to identify name and age; the age is number and every name start with uppercase therefore with regular expression we can find out two digit number and words that start with upper case letter and then goes on with lower case letters. for that the regular expression should be like this

```
ages = re.findall(r'\d{1,3}', NameAge)
names = re.findall(r'[A-Z][a-z]*', NameAge)
```

Python code to implement the above concepts:

```
import re
NameAge = '''
Raj is 14 and Palak is 15
Viyom is 13 and Raju is 17
'''
ages = re.findall(r'\d{1,3}', NameAge)
#findall return list of all matches
names = re.findall(r'[A-Z][a-z]*', NameAge)
ageDict = {}

x=0
for eachName  in names:
        ageDict[eachName] = ages[x]
        x+=1
print(ageDict)
```

## Matching Characters

Most letters and characters will simply match themselves. For example, the regular expression test will match the string test exactly.

There are exceptions to this rule; some characters are special metacharacters, and don't match themselves. Instead, they signal that some out-of-the-ordinary thing should be matched, or they affect other portions of the RE by repeating them or changing their meaning.

Here's a complete list of the metacharacters;

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

The first metacharacters we'll look at are `[` and `]`. They're used for specifying a character class, which is a set of characters that you wish to match. Characters can be listed individually, or a range of characters can be indicated by giving two characters and separating them by a `'-'`. For example, `[abc]` will match any of the characters `a`, `b`, or `c`; this is the same as `[a-c]`, which uses a range to express the same set of characters. If you wanted to match only lowercase letters, your RE would be `[a-z]`.

Metacharacters are not active inside classes. For example, `[akm$]` will match any of the characters `'a'`, `'k'`, `'m'`, or `'$'`; `'$'` is usually a metacharacter, **but inside a character class it's stripped of its special nature.**

You can match the characters not listed within the class by *complementing* the set. This is indicated by including a `'^'` as the first character of the class; `'^'` outside a character class will simply match the `'^'` character. For example, `[^5]` will match any character except `'5'`.

Some of the special sequences beginning with `'\'` represent predefined sets of characters that are often useful, such as the set of digits, the set of letters, or the set of anything that isn't whitespace.

| | |
|---|---|
| `\d` | Matches any decimal digit; this is equivalent to the class `[0-9]`. |
| `\D` | Matches any non-digit character; this is equivalent to the class `[^0-9]`. |
| `\s` | Matches any whitespace character; this is equivalent to the class `[ \t\n\r\f\v]`. |
| `\S` | Matches any non-whitespace character; this is equivalent to the class `[^ \t\n\r\f\v]`. |
| `\w` | Matches any alphanumeric character; this is equivalent to the class `[a-zA-` |

| | |
|---|---|
| | `Z0-9_]`. |
| `\W` | Matches any non-alphanumeric character; this is equivalent to the class `[^a-zA-Z0-9_]`. |

These sequences can be included inside a character class. For example, `[\s,.]` is a character class that will match any whitespace character, or `','` or `'.'`

Now you can query the match object for information about the matching string. Match object instances also have several methods and attributes; the most important ones are:

| Method/Attribute | Purpose |
|---|---|
| group() | Return the string matched by the RE |
| start() | Return the starting position of the match |
| end() | Return the ending position of the match |
| span() | Return a tuple containing the (start, end) positions of the match |

## Various operations that we can perform with regular expression

**Find a word in a string**

```
# we have a sentence and we have to find a word inform in
#it
import re
if re.search("inform", "we need to inform him with the
latest information"):
        print("There is inform")
```

---

```
# for all inform in the string or sentence
import re
allinforms = re.findall("inform", "we need to inform him
with the latest information")
# findall return list of all matches searches
for i in allinforms:
        print(i)
```

It show the output :

inform

inform

## How to generate an iterator

Get the starting and ending index of a particular string

```
#Getting starting and ending index of particular string
import re
str = "we need to inform him with the latest information"
for i in re.finditer("inform", str):
        loctup = i.span()
        print(loctup)
```

The output is

(11, 17)

(38, 44)

The finditer() method return all the starting and ending index in the form of tuple (11, 17) is the starting index of first match for 'inform' and (38, 44) is the second match for inform in the word information.

## Match word with particular pattern

```
import re
str = "Sat, hat, mat, pat"
# now I want to match anything that end with 'at' for that

allstr = re.findall("[shmp]at", str)

# in the pattern [shmp]at we specify, we need a word that
may #be startwith s h m or p and ends with at

for i in allstr:
        print(i)
```

The Output shows

hat

mat

pat

The output not having a string Sat, because it start with uppercase 'S', if we replace our pattern with `[Shmp]at` then it display

Sat

hat

mat

pat

## Match series of range of characters

Now I want all the word in which the first letter between h-m

```
import re
str = "Sat, hat, mat, pat"
allstr = re.findall("[h-m]at", str)
for i in allstr:
        print(i)
```

It shows
hat
mat
because p comes after m and S comes before h
and everything apart from this range we can use '^' symbol in the expression therefore the expression should be like this:

```
"[^h-m]at"
```

## Replace a string

We have a string and we want to replace a particular word from it.

```
import re
str = "hat rat mat pat"
#now I want to replace string 'rat' with 'food' now I am going to compile regular ex with
# pattern object and pattern object provides additional methods and one of it sub()
regex = re.compile("[r]at")
str = regex.sub("food", str)
#first args is string which is to be replace and second args is the string in which we have to replace
print(str)
```

The output is
hat food mat pat

One more example we want to replace '\n' with space character with a particular string.

```
import re
randstr = '''
```

```
Regular expression
supports in all languages
like Java, Python, Ruby etc'''
#print(randstr)
''' Now it show the output
Regular expression
supports in all languages
like Java, Python, Ruby etc
we want to replace all \n with space'''
regex = re.compile("\n")
randstr = regex.sub(" ", randstr)
print(randstr)
Now it shows the output
Regular expression supports in all languages like Java,
Python, Ruby etc
```

**Matching a single character**

```
import re
str = "12345"
print(re.findall("\d", str))
print("Matches :", len(re.findall("\d", str)))
```

\d matches all digits –
findall() return list of all matches,
The output is :
['1', '2', '3', '4', '5']
Matches : 5
\D return other than a digits, if we put "\D" instead of "\d" then
the output is:
[]
Matches : 0

Now if I want to matches a specific digit then we will use {}, and our regular expression should be "\d{5}" then it matches 5 and no of matches will be 1.
The output will be:
['12345']
Matches : 1

Now even we can match digits with certain range

```
import re
```

```
str = "123 1234 12345 123456 1234567"
print(re.findall("\d{5,7}", str)),
#we can sepcify a range between 5 to 7
print("Matches : ", len(re.findall("\d{5,7}", str)))
```

It find three matches for this range therefore output will be
['12345', '123456', '1234567']
Matches : 3

## Regular expression and its application

i) Phone number verification

```
import re
#verify all phone numbers
#\w[a-zA-Z0-9_]
#\W[^a-zA-Z0-9] other than this
pno = "412-555-1212"
#if re.search("\d{3}-\d{3}-\d{4}", pno):
#       print("It is valid phone no")
# if we replace the above expression then \w{3}-\w{3}-\w{3}
also it will do
if re.search("\w{3}-\w{3}-\w{4}", pno):
      print("It is valid phone no")
```

```
The Output is
It is valid phone no (in case of both expressions)
```

ii) Full Name is valid or not

for that there is a space between first name and last name,

```
import re
# \s [\f\n\r\t\v]
#\S [^\f\n\r\t\v]
fullName = "Vikas Nimje"
if(re.search("\w{2, 20}\s\w{2, 20}", fullName)):
       print("Full Name is valid")
```

```
The output is :
Full Name is valid
```

iii) Email verification

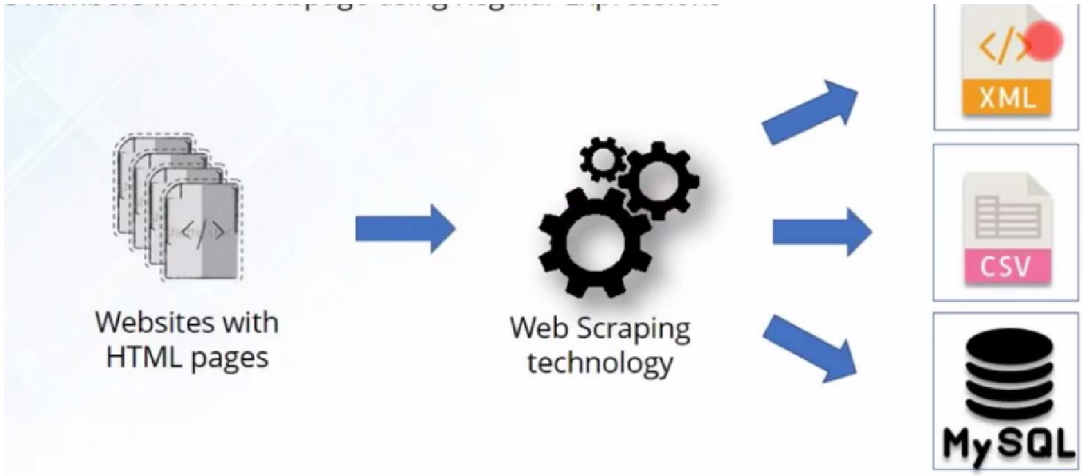
Email address is in the format
is made up of a local-part, an @ symbol, then a case-insensitive domain.

| | |
|---|---|
| Local-part | 1-20 Uppercase and lowercase letters, numbers, plus._%+- |
| An @ symbol | @ symbol |
| Domain Name | 2-20 Uppercase and lowercase letters, numbers, plus, . - |
| A period | . |
| Domain Type | 2 to 3 lowercase and uppercase letters |

```
import re
email = "md@.com @seo.com dc@.com abc@gmail.com"
print(re.findall("[\w._%+-]{1,20}@[\w.-]{2,20}.[A-Za-z]{2,  3}",
email))
print("EmailMatches:",len(re.findall("[\w._%+-]{1,20}@[\w.-
]{2,20}.[A-Za-z]{2, 3}",email)))
```

## Web Scraping using Regular Expression

Scrap all the phone numbers from a webpage using Regular Expressions




The code that find out phone number from name address and phone no from a web page.

```
import urllib.request
from re import findall
url=
"https://www.summet.com/dmsi/html/codesamples/addresses.html"
response = urllib.request.urlopen(url)
html = response.read()
#print(html)
htmlStr = html.decode()
#print(htmlStr)
pdata = findall("\(\d{3}\) \d{3}-\d{4}", htmlStr)
for i in pdata:
       print(i)
```

```
The Output is :
(257) 563-7401
(372) 587-2335
(786) 713-8616
(793) 151-6230
(492) 709-6392
(654) 393-5734
(404) 960-3807
(314) 244-6306
(947) 278-5929
(684) 579-1879
(389) 737-2852
(660) 663-4518
(608) 265-2215
(959) 119-8364
(468) 353-2641
(248) 675-4007
(939) 353-1107
(570) 873-7090
(302) 259-2375
```