Welcome to the lecture notes 15, my name is Akshansh (+91 8384891269, akshansh0fficial@gmail.com). Today we are going to disuss another important topic. We I jump into this, let me tell you this going to be very interesting one.

You all use different platforms like facebook,twitter,instagram,github,stackoverflow and askubuntu (pornhub, pornfidelity for many of us, they are also giving free access to premium so that you all can stay at home during quarantine period, thank me later :-p). What happens when you sign up for these platform? It asks information those are supposed to be valid. My email address is Akshanshofficial@gmail.com, suppose I'd entered akshanshofficial.com, it will raise an error and will tell me that email address entered is invalid. So how do they find that email is invalid?

there is a criteria for what an email can have. So filteration is made in such an order that entered email must match the criteria. If it doesn't, your email is invalid.

# Before starting anything in this Lecture note, make sure you've opened https://regex101.com to practice this.

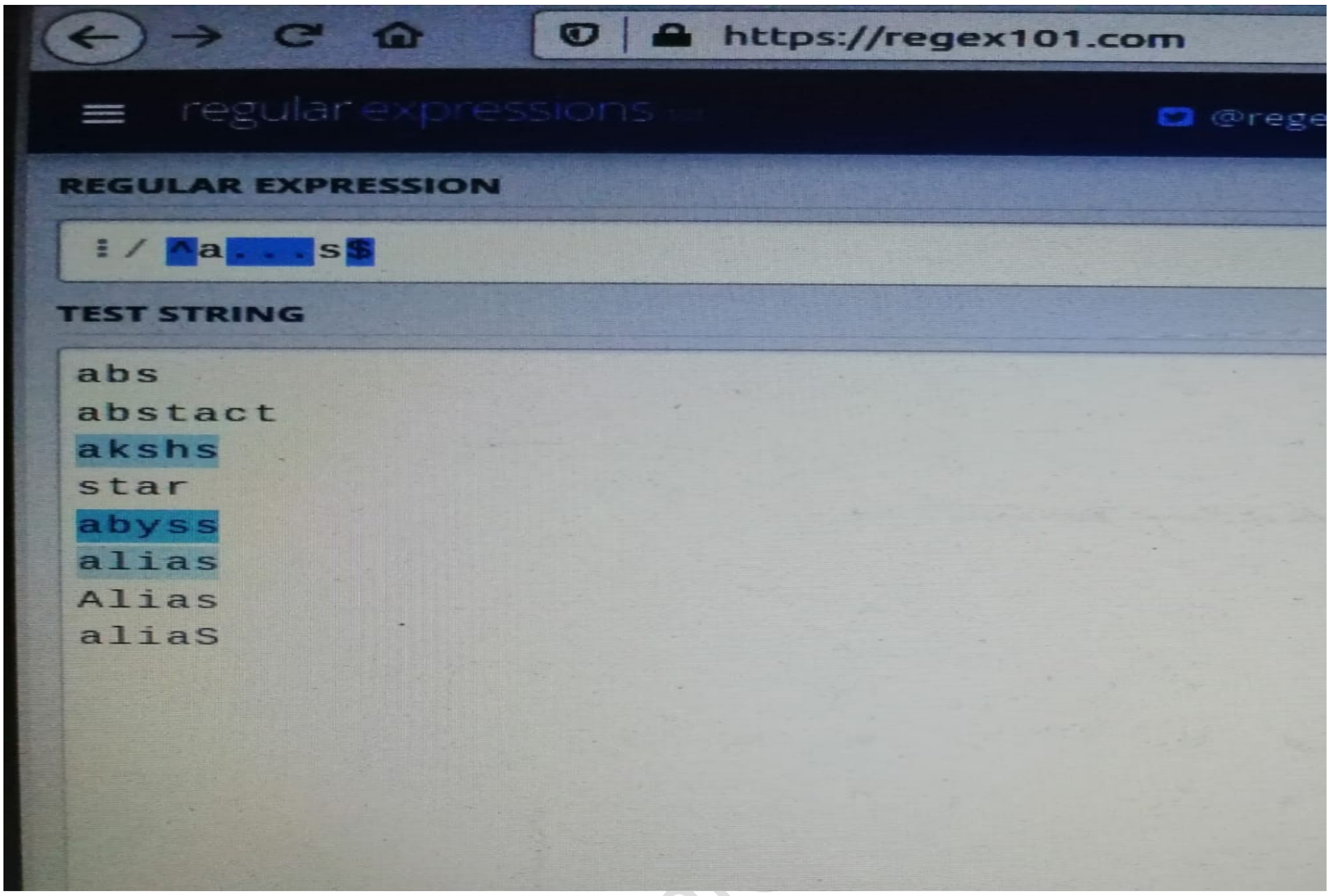
# Python RegEx

Suppose you want to search (a letter or a whole word or you gf) you will need regular expression. Regex will do that for you. Oh forget the gf part there, no one can find it for you. :-P

A regular expression (RegEx) is a sequence of characters that defines a search pattern in your data.

for example : ^a...s$ is a code that defines a regex pattern. Ther pattern is - any five letter string(word) starting with 'a' (not "A") and ending with 's'.

Look at the trial run below -



```
:/ ^a...s$
```

```
abs
abstact
akshs
star
abyss
alias
Alias
aliaS
```

You can see that I've written ^a...s$ in testing search bar and below in test string, I passed multiple strings. What did exactly happen? It looked for the string and the criteria – **criteria was look for the string which is 5 character long and begins with a and ends with s.**

You can do it too, if you are hardcore programmer-

```
import re

pattern = '^a...s$'
test_string = 'abyss'
result = re.match(pattern, test_string)

if result:
  print("Search successful.")
else:
  print("Search unsuccessful.")
```

Specify pattern Using RegEx:

To specify regular expressions, meta-characters are used (what the fcuk is meta character?). In the above example - ^ and $ are meta characters-

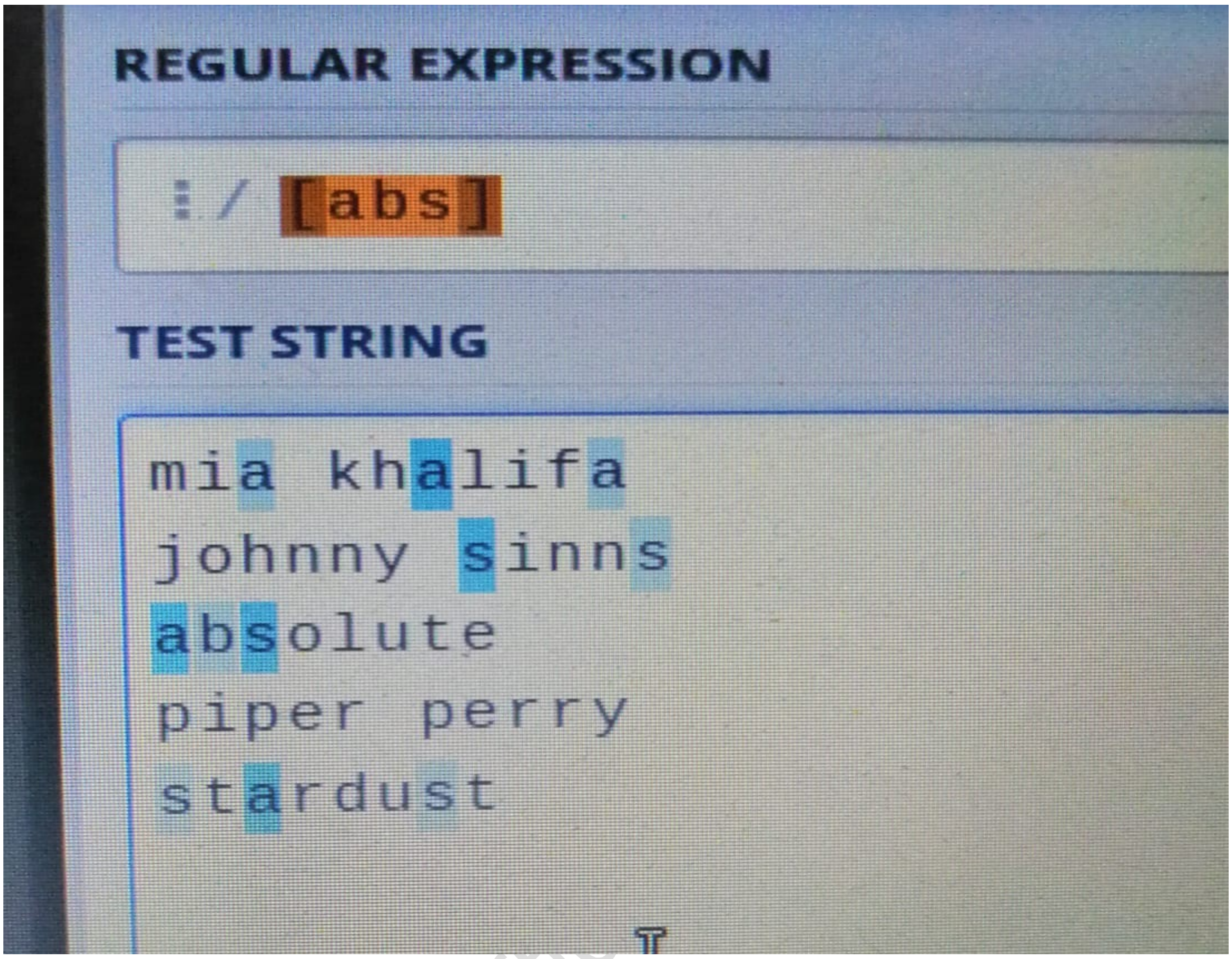
Metacharacters: metacharacters are characters that are interpreted in a special way by a RegEx engine. Here’s a list of meta-characters:

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

Let’s discuss all these one by one – (open your https://regex101.com in pc or mobile, works perfectly on both)

1) []- SQUARE BRACKETS-

Square brackets specifies a set of characters you wish to match-



```
/ [abs]
```

```
mia khalifa
johnny sinns
absolute
piper perry
stardust
```

You can also specify a range of characters using – inside the square brackets.
# [a-e] is same as [abcde]
#[1-4] is same as [1234]

Hope you got it now.

You can complement (invert) the character set by using caret ^ symbol at the start of a square bracket.
# [^abc] means any character except a or b or c
#[^0-9] means non digit characters (all digits comes between 0 to 9)

2) . PERIOD

it matches any single character including space too but it does not find(match a new line '\n' in python)
# . it will match each single character be it alphabet or digits or space
# .. it will match two two character in each set
# ... it will match three three character in each set

(I am not uploading image, try it yourself, you lazy chrocodile)

3) ^ CARET

This is being used to check if a string starts with a certain character.

Expression	String	Matched?
	`a`	1 match
`^a`	`abc`	1 match
	`bac`	No match
`^ab`	`abc`	1 match
	`acb`	No match (starts with `a` but not followed by `b`)

Note in the last string, it didn't match because, it started with 'a' correct but as per the expression 'b' must be right after 'a', which is not in this case. 'c' is right after a so it didn't match. Hope you got it.

4) $ DOLLOR

This is used to check if a strings ends with a certain character.

Expression	String	Matched?
	`a`	1 match
`a$`	`formula`	1 match
	`cab`	No match

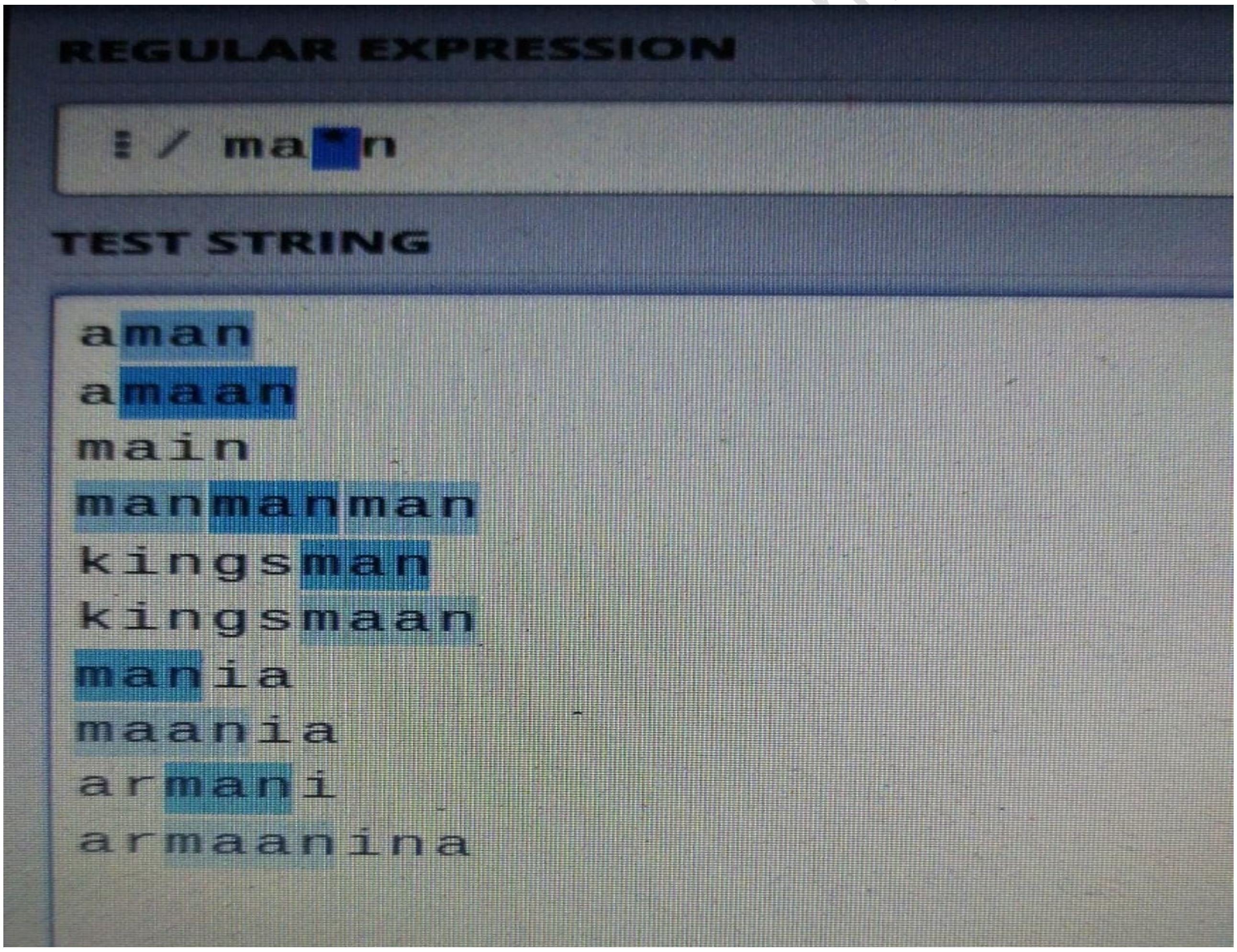
Cab didn't match the criteria, it didn't end with 'a' .

5) * STAR

This is being used to match zero or more occurances of the pattern left to it.

#ma*n this means look for the patterns which starts from ‘m’ and ends at ‘n’. You can see that ma*n, left side of ‘n’ there is ‘a’. It must come zero or many times. If there is no ‘a’ it will match, and if there is more than 1 ‘a’ it will still match. The only condition is, it must be between m and n

Expression	**String**	**Matched?**
	`mn`	1 match
	`man`	1 match
`ma*n`	`maaan`	1 match
	`main`	No match (`a` is not followed by `n`)
	`woman`	1 match


REGULAR EXPRESSION

```
:/ ma*n
```

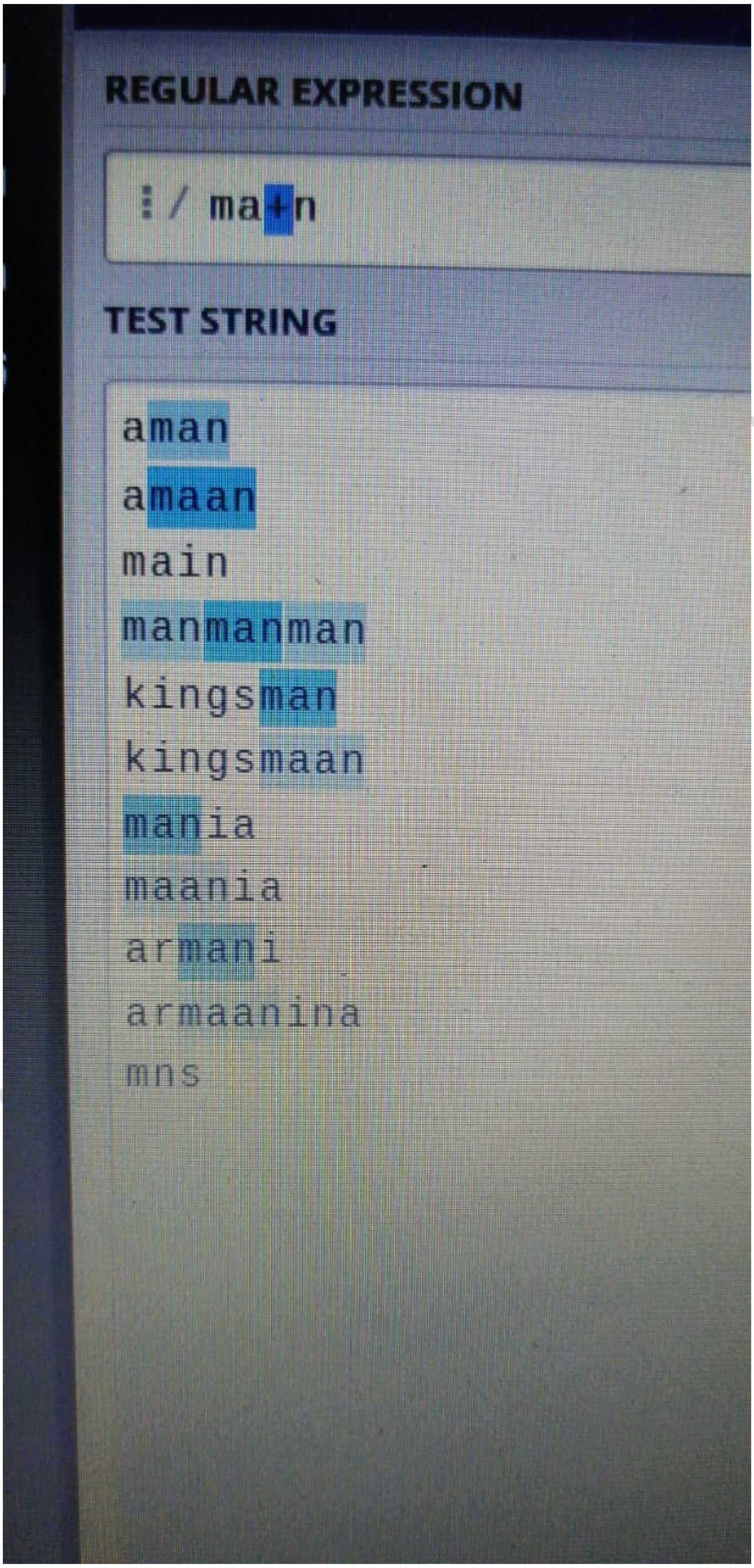
TEST STRING

```
aman
amaan
main
manmanman
kingsman
kingsmaan
mania
maania
armani
armaanina
```

Try mns in that https://reg101.com

6) + PLUS

This means one or more than one occurrence pattern left to it.
#ma+n means look for the pattern those start with m and ends with n. Between those pattern ‘a’ must be atlease one time. If more than once, its okay.



REGULAR EXPRESSION

```
/ ma+n
```

TEST STRING

```
aman
amaan
main
manmanman
kingsman
kingsmaan
mania
maania
armani
armaanina
mns
```

7) ? QUESTION MARK

this looks that there should be one or zero occurrence of the pattern left to it, not more than that.

#ma?n means look for the pattern starting with m and ends with n, between this pattern a can occurr one or zero time. More than one ‘a’ is not allowed.

(No image, try by your self, https://reg101.com is waiting for you.)

8) {} BRACES

consider this code {n,m}. This meanst something must be atleast ‘n’ times and atmost ‘m’ times repeated in string

#a{2,3} means, a must be minimun 2 times or maximum 3 times, it will match those.

Expression	String	Matched?
	`abc dat`	No match
`a{2,3}`	`abc daat`	1 match (at `daat`)
	`aabc daaat`	2 matches (at `aabc` and `daaat`)
	`aabc daaaat`	2 matches (at `aabc` and `daaaat`)

#lets try this, [0-9]{2,4}

what does that means it consits two metacharacters, square brackets and braces, square brackets measn look for the the numbers between 0 to 9 and braces means it (any digit between 0-9) must be atleast 2 times but not more than 4 times.

Expression	String	Matched?
	`ab123csde`	1 match (match at `ab123csde`)
`[0-9]{2,4}`	`12 and 345673`	2 matches (at `12 and 345673`)
	`1 and 2 _`	No match

9) | ALTERNATION

this is being used as choice
# a|b means , either a or b must be there or both can be.

Expression	String	Matched?
	`cde`	No match
`a|b`	`ade`	1 match (match at `ade`)
	`acdbea`	3 matches (at `acdbea`)

10) () GROUP OR BRACKETS

Parenthese is used to group sub patterns.

(a|b|c)xz means, either a or b or c, followed by xz

Expression	String	Matched?
	`ab xz`	No match
`(a|b|c)xz`	`abxz`	1 match (match at `abxz`)
	`axz cabxz`	2 matches (at `axzbc cabxz`)

11) \ BACKSLASH

backslash is used to escape various characters including all metacharacters. Simply saying, you can not use meta character directly in regex search engine because each meta character has a special meaning of. Suppose if you want to search $ in your string, you won’t be able to search it because it is a meta character. So how you find it? Yes, \ helps you.

\$a means look for the strings in whish $ is followed by ‘a’

SPECIAL SEQUENCES

12) \A means look for the characters if they are at the begining of the string.

Expression	String	Matched?
`\Athe`	`the sun`	Match
	`In the sun`	No match

13) \b means the characters are at the start of the string or at the end of the string.

Expression	String	Matched?
	`football`	Match
`\bfoo`	`a football`	Match
	`afootball`	No match
	`the foo`	Match
`foo\b`	`the afoo test`	Match
	`the afootest`	No match

14) \B means, just opposite to \b. It measn character must not be at the start and at the end.

Expression	String	Matched?
	`football`	No match
`\Bfoo`	`a football`	No match
	`afootball`	Match
	`the foo`	No match
`foo\B`	`the afoo test`	No match
	`the afootest`	Match

15) \d means look for the digits, which is equivalent of [0-9]

Expression	String	Matched?
`\d`	`12abc3`	3 matches (at `12abc3`)
	`Python`	No match

16) \D means opposite to \d that means there should not be a digits. Equivalent of [^0-9]

Expression	String	Matched?
`\D`	`1ab34"50`	3 matches (at `1ab34"50`)
	`1345`	No match

17) \s matches where a string contains any whitespace character. Equivalent of [\t\n\r\f\v]

Expression	String	Matched?
`\s`	`Python RegEx`	1 match
	`PythonRegEx`	No match

18) \S opposite to \s, look foe everything other than space character

Expression	String	Matched?
`\S`	`a b`	2 matches (at `a b`)
		No match

19) \w finds any alphanumeric character (digits and alphabets), equivalent [a-zA-Z0-9_], note that it also finds underscores too.

Expression	String	Matched?
`\w`	`12&": ;c`	3 matches (at `12&": ;c`)
	`%"> !`	No match

20) \W opposite to \w

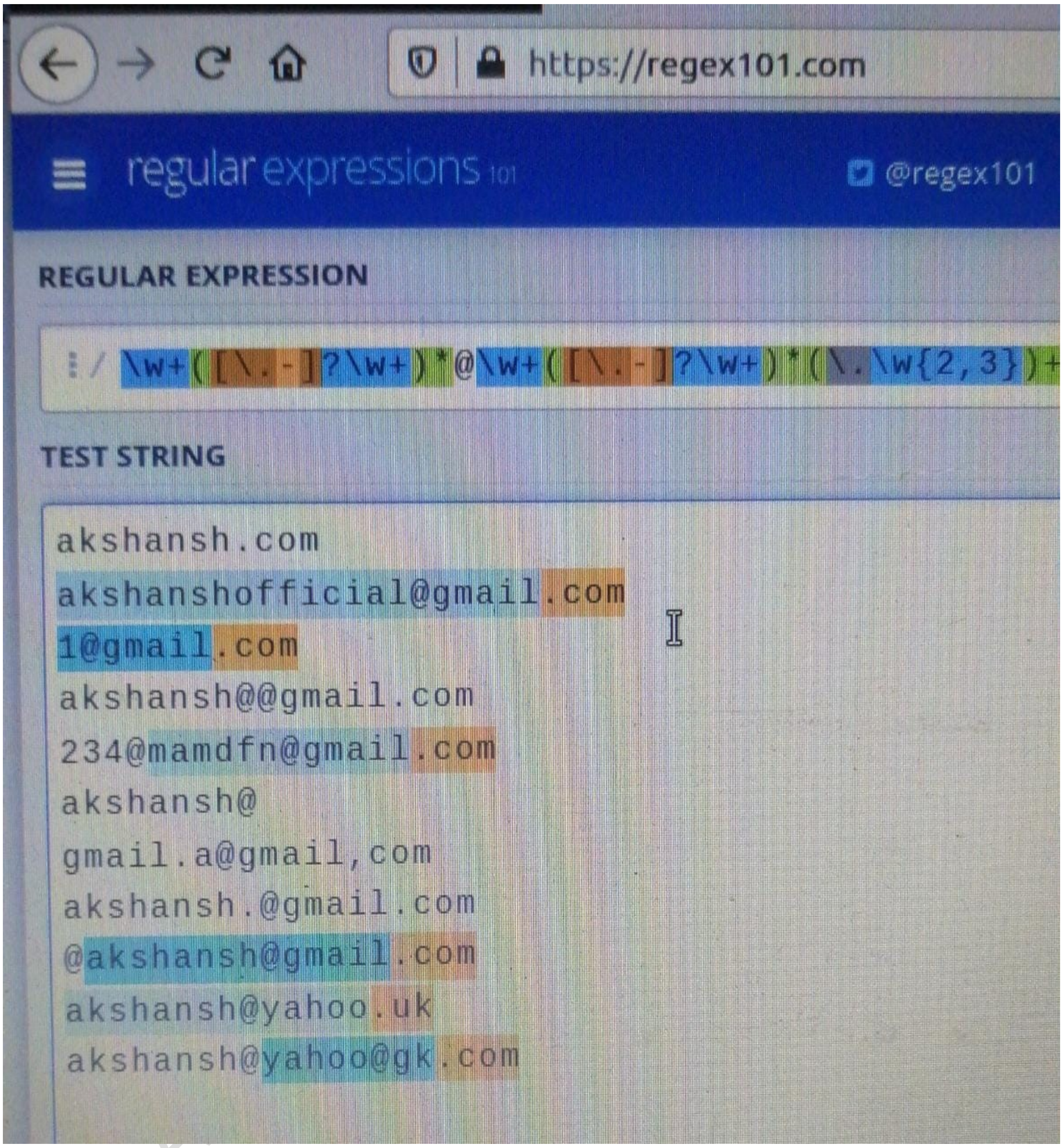
21) \Z Matches if the specified characters are at the end of the string.

Expression	String	Matched?
	`I like Python`	1 match
`\ZPython`	`I like Python`	No match
	`Python is fun.`	No match

You must be wondering how to validate email, I told you earlier about. Well you’ve to know how to play with expressions in this case.

Let me drop a hint :

```
\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$
```



```
/ \w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+
```

```
akshansh.com
akshanshofficial@gmail.com
1@gmail.com
akshansh@@gmail.com
234@mamdfn@gmail.com
akshansh@
gmail.a@gmail,com
akshansh.@gmail.com
@akshansh@gmail.com
akshansh@yahoo.uk
akshansh@yahoo@gk.com
```

Python RegEx

Python has a module named `re` to work with regular expressions. To use it, we need to import the module.

```
import re
```

The module defines several functions and constants to work with RegEx.

re.findall()

The `re.findall()` method returns a list of strings containing all matches.

Example 1: re.findall()

```
# Program to extract numbers from a string

import re

string = 'hello 12 hi 89. Howdy 34'
pattern = '\d+'

result = re.findall(pattern, string)
print(result)

# Output: ['12', '89', '34']
```

If the pattern is no found, `re.findall()` returns an empty list.

---

re.split()

The `re.split` method splits the string where there is a match and returns a list of strings where the splits have occurred.

Example 2: re.split()

```
import re

string = 'Twelve:12 Eighty nine:89.'
pattern = '\d+'

result = re.split(pattern, string)
print(result)

# Output: ['Twelve:', ' Eighty nine:', '.']
```

If the pattern is no found, `re.split()` returns a list containing an empty string.

You can pass `maxsplit` argument to the `re.split()` method. It's the maximum number of splits that will occur.

```
import re

string = 'Twelve:12 Eighty nine:89 Nine:9.'
pattern = '\d+'
```

```
# maxsplit = 1
# split only at the first occurrence
result = re.split(pattern, string, 1)
print(result)

# Output: ['Twelve:', ' Eighty nine:89 Nine:9.']
```

By the way, the default value of `maxsplit` is 0; meaning all possible splits.

---

re.sub()

The syntax of `re.sub()` is:

```
re.sub(pattern, replace, string)
```

The method returns a string where matched occurrences are replaced with the content of *replace* variable.

Example 3: re.sub()

```
# Program to remove all whitespaces
import re

# multiline string
string = 'abc 12\
de 23 \n f45 6'

# matches all whitespace characters
pattern = '\s+'

# empty string
replace = ''

new_string = re.sub(pattern, replace, string)
print(new_string)

# Output: abc12de23f456
```

If the pattern is no found, `re.sub()` returns the original string.

You can pass *count* as a fourth parameter to the `re.sub()` method. If omited, it results to 0. This will replace all occurrences.

```
import re

# multiline string
string = 'abc 12\
de 23 \n f45 6'

# matches all whitespace characters
pattern = '\s+'
```

```
replace = ''

new_string = re.sub(r'\s+', replace, string, 1)
print(new_string)

# Output:
# abc12de 23
# f45 6
```

---

re.subn()

The `re.subn()` is similar to `re.sub()` expect it returns a tuple of 2 items containing the new string and the number of substitutions made.

Example 4: re.subn()

```
# Program to remove all whitespaces
import re

# multiline string
string = 'abc 12\
de 23 \n f45 6'

# matches all whitespace characters
pattern = '\s+'

# empty string
replace = ''

new_string = re.subn(pattern, replace, string)
print(new_string)

# Output: ('abc12de23f456', 4)
```

---

re.search()

The `re.search()` method takes two arguments: a pattern and a string. The method looks for the first location where the RegEx pattern produces a match with the string.

If the search is successful, `re.search()` returns a match object; if not, it returns `None`.

```
match = re.search(pattern, str)
```

Example 5: re.search()

```
import re

string = "Python is fun"
```

```
# check if 'Python' is at the beginning
match = re.search('\APython', string)

if match:
  print("pattern found inside the string")
else:
  print("pattern not found")

# Output: pattern found inside the string
```

Here, *match* contains a match object.

Match object

You can get methods and attributes of a match object using dir() function.

Some of the commonly used methods and attributes of match objects are:

---

match.group()

The `group()` method returns the part of the string where there is a match.

Example 6: Match object

```
import re

string = '39801 356, 2102 1111'

# Three digit number followed by space followed by two digit number
pattern = '(\d{3}) (\d{2})'

# match variable contains a Match object.
match = re.search(pattern, string)

if match:
  print(match.group())
else:
  print("pattern not found")

# Output: 801 35
```

Here, *match* variable contains a match object.

Our pattern `(\d{3}) (\d{2})` has two subgroups `(\d{3})` and `(\d{2})`. You can get the part of the string of these parenthesized subgroups. Here's how:

```
>>> match.group(1)
'801'

>>> match.group(2)
'35'
>>> match.group(1, 2)
('801', '35')
```

```
>>> match.groups()
('801', '35')
```

---

match.start(), match.end() and match.span()

The `start()` function returns the index of the start of the matched substring. Similarly, `end()` returns the end index of the matched substring.

```
>>> match.start()
2
>>> match.end()
8
```

The `span()` function returns a tuple containing start and end index of the matched part.

```
>>> match.span()
(2, 8)
```

---

match.re and match.string

The `re` attribute of a matched object returns a regular expression object. Similarly, `string` attribute returns the passed string.

```
>>> match.re
re.compile('(\\d{3}) (\\d{2})')

>>> match.string
'39801 356, 2102 1111'
```

We have covered all commonly used methods defined in the `re` module. If you want to learn more, visit Python 3 re module.

Using r prefix before RegEx

When *r* or *R* prefix is used before a regular expression, it means raw string. For example, `'\n'` is a new line whereas `r'\n'` means two characters: a backslash \ followed by `n`.

Backlash \ is used to escape various characters including all metacharacters. However, using *r* prefix makes \ treat as a normal character.

Example 7: Raw string using r prefix

```
import re

string = '\n and \r are escape sequences.'
```

```
result = re.findall(r'[\n\r]', string)
print(result)

# Output: ['\n', '\r']
```

I guess this is one of the longest lecture notes, right. Anyway, I believe in you. This is from my end, regex has been wrapp up so the basic python too. We'll move to Numpy and pandas now. This was akshansh and it was my pleasure to teach you python basics. Your feedback is essentially required, please drop your feedback at akshanshofficial@gmail.com , it will help me to teach more efficiently. Thanks to all, all the support in group, you tube and thread email has been great from you guys. Some of us will part our ways now, I wish you all of the best and I hope you'll get success in whatever you are up to. My time is up now. Stay safe from COVID_19 and stay at home.

Have a great day, all.

Thanks & regards,
Akshansh, ||+91 8384891269, akshanshofficial@gmail.com||