

LECTURE 18

Regular expressions

Regular Expression E-mail Matching Example

`/[\w._%+-]+@[\w.-]+\.[a-zA-Z]{2,4}/`

REGULAR EXPRESSIONS (RE, REGEX)

- Notation for describing **simple** string patterns
- Very useful for text processing:
 - Finding/extracting patterns in the text
 - Manipulating strings
 - Checking string correctness

SEARCH IN A LOG FILE FOR A DATA OR/AND TIME?

```
Catalogue listing of /Users/hoakley/Documents/0newDownloads/logstuff/0logarchive/dest/Full1.logarchive :
Persist/00000000000000268.tracev3 2017-09-03 14:43:19 2017-09-04 09:50:22 10477040 bytes, period 1147.0 min
Persist/00000000000000269.tracev3 2017-09-04 09:52:11 2017-09-04 20:42:44 10483320 bytes, period 650.5 min
Persist/0000000000000026a.tracev3 2017-09-04 20:42:58 2017-09-05 13:31:30 10478896 bytes, period 1008.5 min
Persist/0000000000000026b.tracev3 2017-09-05 13:31:30 2017-09-05 22:06:04 10480248 bytes, period 514.6 min
Persist/0000000000000026c.tracev3 2017-09-05 22:06:04 2017-09-06 16:17:08 10473488 bytes, period 1091.1 min
Persist/0000000000000026d.tracev3 2017-09-06 16:21:19 2017-09-07 08:58:59 10473696 bytes, period 997.7 min
Persist/0000000000000026e.tracev3 2017-09-07 08:58:59 2017-09-07 17:59:08 10483432 bytes, period 540.1 min
Persist/0000000000000026f.tracev3 2017-09-07 18:00:46 2017-09-08 09:49:59 10482224 bytes, period 949.2 min
Persist/00000000000000270.tracev3 2017-09-08 09:50:00 2017-09-08 22:03:47 10482224 bytes, period 733.8 min
Persist/00000000000000271.tracev3 2017-09-08 22:03:48 2017-09-09 17:48:56 10483488 bytes, period 1185.1 min
Persist/00000000000000272.tracev3 2017-09-09 17:49:13 2017-09-10 10:23:52 10472576 bytes, period 994.6 min
Persist/00000000000000273.tracev3 2017-09-10 10:24:16 2017-09-11 05:35:53 10482240 bytes, period 1151.6 min
Persist/00000000000000274.tracev3 2017-09-11 05:35:53 2017-09-11 20:39:47 10486576 bytes, period 903.9 min
Persist/00000000000000275.tracev3 2017-09-11 20:39:52 2017-09-12 15:20:50 10477104 bytes, period 1121.0 min
Persist/00000000000000276.tracev3 2017-09-12 15:20:50 2017-09-13 07:11:04 10475936 bytes, period 950.2 min
Persist/00000000000000277.tracev3 2017-09-13 07:11:29 2017-09-13 19:32:24 10483424 bytes, period 740.9 min
Persist/00000000000000278.tracev3 2017-09-13 19:32:27 2017-09-14 14:36:05 10482800 bytes, period 1143.6 min
Persist/00000000000000279.tracev3 2017-09-14 14:36:06 2017-09-15 05:22:03 10481936 bytes, period 886.0 min
Persist/0000000000000027a.tracev3 2017-09-15 05:22:03 2017-09-15 17:59:43 10476536 bytes, period 757.7 min
Persist/0000000000000027b.tracev3 2017-09-15 17:59:48 2017-09-16 06:23:12 10485344 bytes, period 743.4 min
Persist/0000000000000027c.tracev3 2017-09-16 06:23:14 2017-09-16 22:16:08 10481768 bytes, period 952.9 min
Persist/0000000000000027d.tracev3 2017-09-16 22:16:09 2017-09-17 07:38:38 1542408 bytes, period 562.5 min
```

FAST VERIFICATION OF

- Address
- Email address
- Time
- Passwords
- Names
- Credit Card numbers
- ...

REGULAR EXPRESSIONS CAN BE USED ANYWHERE!!

- Python
- SQL
- Java (and many many other programming languages)
- Text Editors (Atom, Notepad ++, etc)
- IDEs (Eclipse, IntelliJ, etc)
- Tableau
- ...

EXAMPLE. WHAT WOULD YOU DO?

Find all numbers in the given text

```
line = "2 plus 2 is 4 and not 567".
```

EXAMPLE. SOLUTION WITHOUT RE

Find all numbers in the given text

```
line = "2 plus 2 is 4 and not 567".
```

```
>>> line = "2 plus 2 is 4 and not 567"
```

```
>>> [int(s) for s in line.split() if s.isdigit()]
```

```
[2, 2, 4, 567]
```


EXAMPLE

```
import re
```

```
line = "2 plus 2 is 4 and not 567"
```

```
matched = re.findall('[0-9]+', line)
```

```
print(matched)
```

EXAMPLE. WHAT IS THE PROBLEM?

```
import re
```

```
line = "2 plus 2 is 4 and not 567"
```

```
matched = re.findall('[0-9]+', line)
```

```
print(matched)
```

```
['2', '2', '4', '567']
```

EXAMPLE. FIX

```
import re
```

```
line = "2 plus 2 is 4 and not 567"
```

```
matched = re.findall('[0-9]+', line)
```

```
int_list = [int(s) for s in matched]
```

```
print(int_list)
```

```
[2, 2, 4, 567]
```

EXAMPLE. ISSUE? SOMETIMES..

```
import re
```

```
line = "Aqua34, 2 plus 2 is 4 and not 567"
```

```
matched = re.findall('[0-9]+', line)
```

```
int_list = [int(s) for s in matched]
```

```
print(int_list)
```

```
[34, 2, 2, 4, 567]
```

EXAMPLE. ISSUE? FIX.

```
import re
```

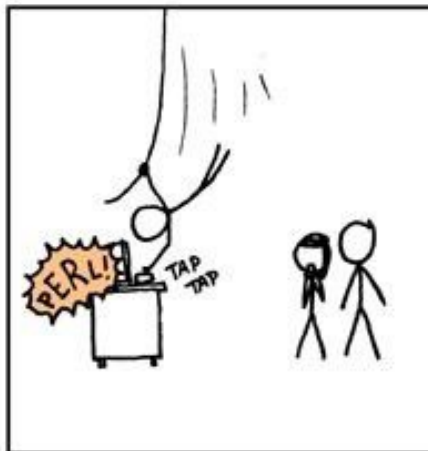
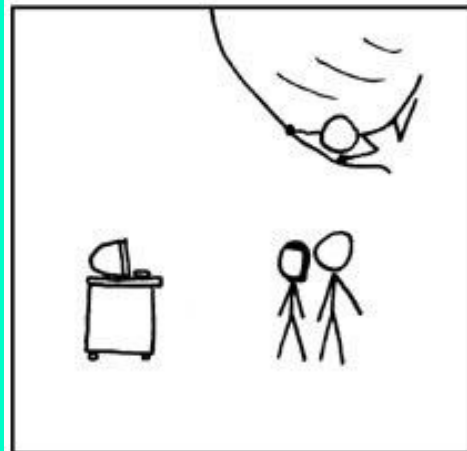
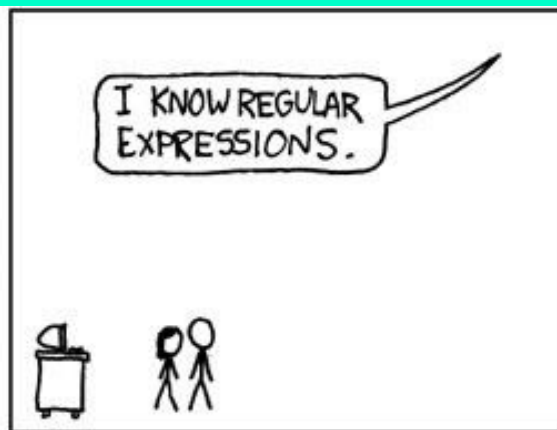
```
line = "Aqua34, 2 plus 2 is 4 and not 567"
```

```
matched = re.findall(r'\b[0-9]+\b', line)
```

```
int_list = [int(s) for s in matched]
```

```
print(int_list)
```

```
[2, 2, 4, 567]
```



REGULAR EXPRESSION (RE, REGEX)

- Special text string for describing a search pattern
 - Mini language
- When apply a RE to a text, usually we get a class of strings (matches)
- Work over some alphabet
 - Letters (small case, upper case, both, numbers, all characters etc)
- String:
 - Can be empty (ε - epsilon)
 - Any sequence of characters over alphabet of length 1 or more

BUILDING BLOCKS, IDENTIFIERS

- `\d` : any digit
- `\D` : anything but a number
- `\s` : space [`\t``\n``\r``\f``\v`]
 - `\f` page break; `\v` vertical tab character
- `\S` : anything but a space
- `\t` : tab
- `\n` : new line
- `\w` : word character: a-z, A-Z, 0-9, including the `_`
- `\W` : anything but a character
- `.` (period) **wildcard**. matches *any* symbol (except new line)
- `\b` : word boundary
- `\.` : A period

BUILDING BLOCKS. MODIFIERS

- `{ }` : `{2, 5}`. `\d{2,5}` Looking for 2-5 digits in a row
- `+` : match 1 or more.
- `?` : match 0 or 1.
- `*` : match 0 or more.
- `$` : match the end of the string
- `^` : match the beginning of the string
- `|` : either/or `\d{2, 5} | \d{5, 8}`
- `[]` : Used to construct character classes (range)
 - `[A - Z]`, `[A-Za-z]`
- `{n}` : Expected “n” amount
- `^` : Negating character class

EXAMPLES. DEMO

```
re.findall(pattern, string)
```

- Extract names
- Extract ages
- Put them in the dictionary

RE.SEARCH (DEMO)

- Search in the text for the *first* occurrence. Return None if nothing is found. Returns the found object otherwise.

MATCH WORDS WITH PARTICULAR PATTERN

`input_string = "Marina, Katya, Irina, Maria, Olga, Bogdan".`

What these words have in common, except the one?

MATCH WORDS WITH PARTICULAR PATTERN

```
input_string = "Marina, Katya, Irina, Maria, Olga, Bogdan".
```

What these words have in common, except the one?

All of them but the last one, ends with an 'a'.

How to find them? (demo)



^ SYMBOL (CARET)

[5]: find only 5

[^5]: find anything but a 5

[^a-t]: all letters but not in a range a-t

A FEW KEY OPERATIONS FOR []

- [ACE]
 - any of the symbols A, C, or E
- [A-E]
 - any of the symbols A, B, C, D, or E
- [A-E_] 
 - any of A-E, or the underscore _
- [^A-E_] 
 - any character not in the previous class (^ negates a class)
- [-A-Z]
 - any of A-Z, or the hyphen
- Using parentheses to change precedence:
 - ABC? matches AB or ABC
 - A(BC)? matches A or ABC

PRACTICE 1

- How many numbers do I have in my text?
 - Using regex, not your eyes :)

```
input = "One, 2, three, 3, 4, 5. Time 4you 2go home. Joke,  
in 50 minutes."
```


PRACTICE 1. SOLUTION

```
input = "One, 2, three, 3, 4, 5. Time 4you 2go home. Joke,  
in 50 minutes."
```

```
all_matches = re.findall(r'\b\d+\b', input)  
print(all_matches)  
  
print(len(all_matches))
```

PRACTICE 1. SOLUTION

```
input = "One, 2, three, 3, 4, 5. Time 4you 2go home. Joke,  
in 50 minutes."
```

```
all_matches = re.findall(r'\b\d+\b', input)    # \b[0-9]+\b ok too  
print(all_matches)  
  
print(len(all_matches))
```

PRACTICE 2, THEN DEMO

```
input = "1, 12, 123, 12345, 123456, 1234567, 12345678"
```

```
all_matches = re.findall(r'\d{3}', input)
```

```
print(len(all_matches))
```

A: 1

D: 8

B: 5

E: Have no idea

C: 7

PRACTICE 2.2 GREEDY MATCHING

```
input = "1, 12, 123, 12345, 123456, 1234567, 12345678"
```

```
all_matches = re.findall(r'\d{3,5}', input)
```

```
print(len(all_matches))
```

A: 1

D: 8

B: 5

E: None of the above

C: 7

PRACTICE 3

Use regular expression to match a 10-digit phone number with dashes and parentheses:

(847)812-4567 : valid

(312)345-7512 : valid

(345)5435674 : invalid

234-534-6434 : invalid

PRACTICE 3.SOLUTION

(847)812-4567 : valid

(312)345-7512 : valid

(345)5435674 : invalid

234-534-6434 : invalid

```
input = "(847)812-4567, (312)345-7512, (345)5435674, 234-534-6434"
```

```
all_matches = re.findall(r'\(\d{3}\)\d{3}-\d{4}', input)
```

PRACTICE 4

Now using '?' make a dash optional in your phone number RE.

(847)812-4567 : valid

(312)345-7512 : valid

(345)5435674 : ~~invalid~~ valid

234-534-6434 : invalid

```
all_matches = re.findall(r'\((\d{3})\)\d{3}\-\d{4}', input)
```

PRACTICE 4. SOLUTION

Now using '?' make a dash optional in your phone number RE.

(847)812-4567 : valid

(312)345-7512 : valid

(345)5435674 : ~~invalid~~ valid

234-534-6434 : invalid

```
all_matches = re.findall(r'\((\d{3})\)\d{3}\-?\d{4}', input)
```


WILDCARDS

`.` (dot): A dot `.` is a replacement for **any** character (if used outside `[]`)

`.*` : which allows 0 or more repetitions of any character.
This is often used to match *any* text

PRACTICE 5.

| : (Pipe, Alternation, OR)

It alternates two or more valid patterns where at least one of those patterns must match in that position.

PRACTICE 5.

| : (Pipe, Alternation, OR)

It alternates two or more valid patterns where at least one of those patterns must match in that position.

Write RE to capture 5-digit U.S. ZIP codes that end in "22" or "30," .

Then write code that checks if a given zip is correct (search)

PRACTICE 5. SOLUTION.

Write RE to capture 5-digit U.S.
ZIP codes that end in "22" or "30,"

```
input_zip = "12311, 2222, 53430, 92122,  
92130, 34530, 30322, 435322, 43225"
```

```
zipSplitted = input_zip.split(',')
```

```
for z in zipSplitted:  
    if re.search(r'\b[0-9]{3}(22|30)\b', z):  
        print(z + " is correct")  
    else:  
        print(z + " is not correct")
```

PRACTICE 6. VALID NAME

Check that a given name (Last and First) has a proper format:

Langlois, Marina: OK

Langlois Marina: OK

Langlois Marina: NOT OK

LangloisMarina: NOT OK

PRACTICE 6. VALID NAME

Check that a given name (Last and First) has a proper format:

Langlois, Marina: OK

Langlois Marina: OK

Langlois Marina: NOT OK

LangloisMarina: NOT OK

```
name = "Langlois, Marina"
```

```
if re.search(r'\w+,? \w+', name):
```

```
    print(name + " is correct")
```

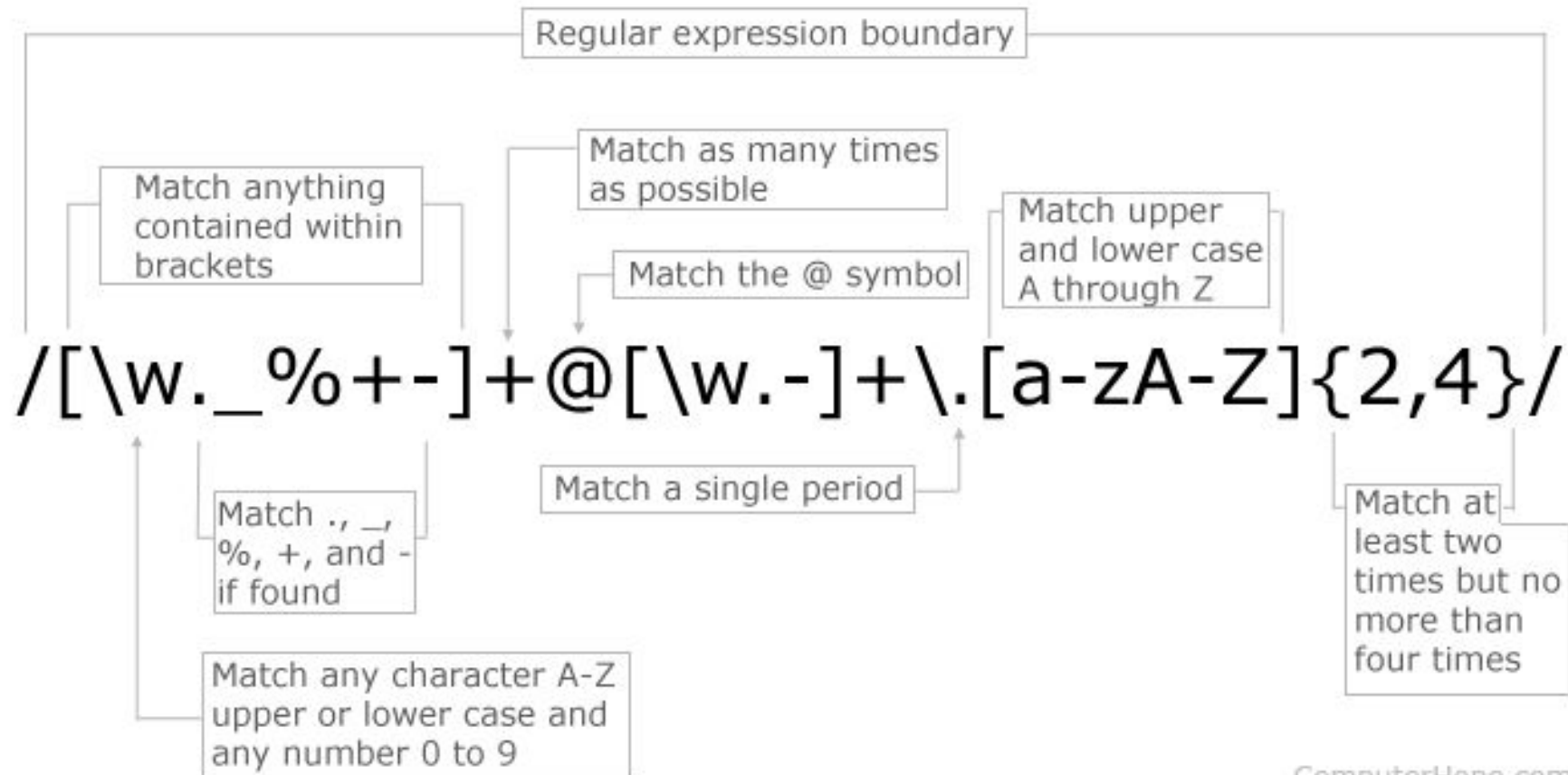
```
else:
```

```
    print(name + " is not correct")
```

Regular Expression E-mail Matching Example

`/[\w._%+-]+@[\w.-]+\.[a-zA-Z]{2,4}/`

Regular Expression E-mail Matching Example



SEARCH AND GROUP

- `re.search()`. This method either returns `None` if the pattern doesn't match, or a `re.MatchObject` with additional information about which part of the string the match was found.
- Note that this method **stops** after the first match, so this is best suited for testing a regular expression more than extracting data.

SEARCH AND GROUP

```
import re
regex = r"([a-zA-Z]+) (\d+)"
match = re.search(regex, "June 24")
if match:
    # If we want, we can use the
    # MatchObject's start() and end()
    # methods to retrieve where the
    # pattern matches in the input string
```

SEARCH AND GROUP

```
import re
regex = r"([a-zA-Z]+) (\d+)"
match = re.search(regex, "June 24")
if match:
    # If we want, we can use the
    # MatchObject's start() and end()
    # methods to retrieve where the
    # pattern matches in the input string

    print("Match at index %s, %s" % (match.start(), match.end()))
```

SEARCH AND GROUP

```
import re
regex = r"([a-zA-Z]+) (\d+)"
match = re.search(regex, "June 24")
if match:
    # If we want, we can use the
    # MatchObject's start() and end()
    # methods to retrieve where the
    # pattern matches in the input string

    print("Match at index %s, %s" % (match.start(), match.end()))
    Match at index 0, 7      ← output

    # This will print [0, 7), since it matches at the beginning and end of
    # the string
```

SEARCH AND GROUP

```
import re
```

```
regex = r"([a-zA-Z]+) (\d+)"
```

```
match = re.search(regex, "June 24")
```

```
if match:
```

```
    # Or we can use a group() method to get all the matches and captured groups.
```

SEARCH AND GROUP

```
import re
regex = r"([a-zA-Z]+) (\d+)"
match = re.search(regex, "June 24")
if match:
```

```
    # Or we can use a group() method to get all the matches and captured groups.
```

```
# The groups contain the matched values. In particular:
```

```
#     match.group(0) always returns the fully matched string
```

```
#     match.group(1), match.group(2), ... will return the capture
```

```
#         groups in order from left to right in the input string
```

```
#     match.group() is equivalent to match.group(0)
```

SEARCH AND GROUP

```
import re
regex = r"([a-zA-Z]+) (\d+)"
match = re.search(regex, "June 24")
if match:
    # Or we can use a group() method to get all the matches and captured groups.
    # The groups contain the matched values. In particular:
    # match.group(0) always returns the fully matched string

    print(match.group(0))
    June 24    <- output
```

SEARCH AND GROUP

```
import re
regex = r"([a-zA-Z]+) (\d+)"
match = re.search(regex, "June 24")
if match:
    # Or we can use a group() method to get all the matches and captured groups.
    # The groups contain the matched values. In particular:
    #     match.group(0) always returns the fully matched string

print(match.group(0))
June 24    <- output
#     match.group(1), match.group(2), ... will return the capture
#           groups in order from left to right in the input string
```


SEARCH AND GROUP

```
import re
regex = r"([a-zA-Z]+) (\d+)"
match = re.search(regex, "June 24")
if match:
    # Or we can use a group() method to get all the matches and captured groups.
    # The groups contain the matched values. In particular:
    # match.group(0) always returns the fully matched string

    print(match.group(0))
    June 24 <- output
    # match.group(1), match.group(2), ... will return the capture
    # groups in order from left to right in the input string

    print("Month: %s" % (match.group(1)))
    print("Day: %s" % (match.group(2)))
```

Month: June <- output
Day: 24

SEARCH AND GROUP

```
import re
regex = r"([a-zA-Z]+) (\d+)"
match = re.search(regex, "June 24")
if match:
    # Or we can use a group() method to get all the matches and captured groups.
    # The groups contain the matched values. In particular:
    #     match.group(0) always returns the fully matched string

    print(match.group(0))
    June 24    <- output
    #     match.group(1), match.group(2), ... will return the capture
    #             groups in order from left to right in the input string

    print("Month: %s" % (match.group(1)))    Month: June <- output
    print("Day: %s" % (match.group(2)))      Day: 24
else:
    print("The regex pattern does not match. :(")
```

SEARCH VS MATCH

`search` ⇒ find something anywhere in the string and return a match object.

`match` ⇒ find something at the *beginning* of the string and return a match object.

```
a = "123abc"
```

```
t = re.search("[a-z]+", a)
```

```
if t:
    print(t.group())
else:
    print("no match")
```

```
abc
```

SEARCH VS MATCH

`search` ⇒ find something anywhere in the string and return a match object.

`match` ⇒ find something at the *beginning* of the string and return a match object.

```
a = "123abc"
```

```
t = re.search("[a-z]+", a)
```

```
if t:
    print(t.group())
else:
    print("no match")
```

abc

```
a = "123abc"
```

```
t = re.match("[a-z]+", a)
```

```
if t:
    print(t.group())
else:
    print("no match")
```

no match

PRACTICE

- Write a Python program to separate and print the numbers of a given string.

PRACTICE

- Write a Python program to separate and print the numbers of a given string.

```
import re
# Sample string.
text = "Ten 10, Twenty 20, Thirty
30"
result = re.split("\D+", text)
# Print results.
for element in result:
    print(element)
```

WORD BOUNDARY

- Matches the empty string, but only at the beginning or end of a word
- There are a few rules for "`\b`".
- We know that `\w` matches any alphanumeric character; this is equivalent to the class `[a-zA-Z0-9_]`.
- Also `\W` is the opposite of `\w`

Rules:

- `\b` is defined as the **boundary** between a `\w` and a `\W` character (or vice versa) between `\w` and the beginning/end of the string

WORD BOUNDARY

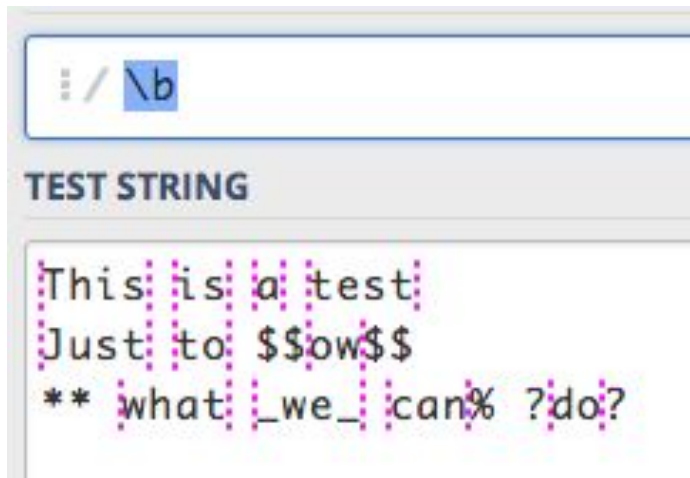
Rules:

- `\b` is defined as the **boundary** between a `\w` and a `\W` character (or vice versa) between `\w` and the beginning/end of the string
- Example:

```
input text = "This is a test
```

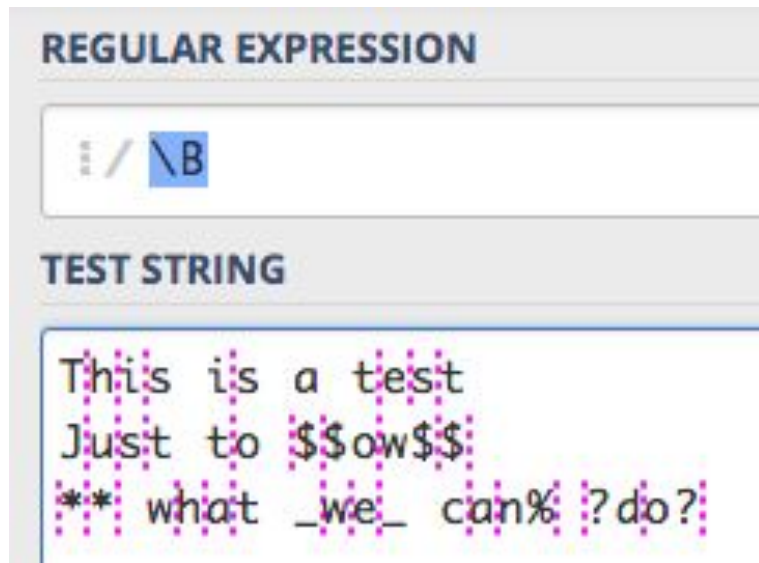
```
Just to $$ow$$
```

```
** what _we_ can% ?do?"
```



WORD BOUNDARY "\B": OPPOSITE TO \b (LITTLE B)

- Matches the empty string, but only when it is not at the beginning or end of a word



HOW TO FIX?

```
input_string = "Marina, Katya, Irina, Maria, Olga, Bogdan".
```

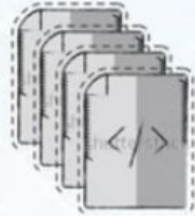
```
Output: ['Marina', 'Katya', 'Irina', 'Maria', 'Olga']
```

```
all_matches2 = re.findall(r'[A-Z][a-z]*a\b', input_line)  
print(all_matches2)
```

WEB SCRAPING (FROM EDURIKA!)

Problem Statement:

Scrap all the phone numbers from a webpage using Regular Expressions



Websites with
HTML pages



Web Scraping
technology



ANNOUNCEMENT

By Samson Qian