SELA

Module 06: Regular expressions

Agenda

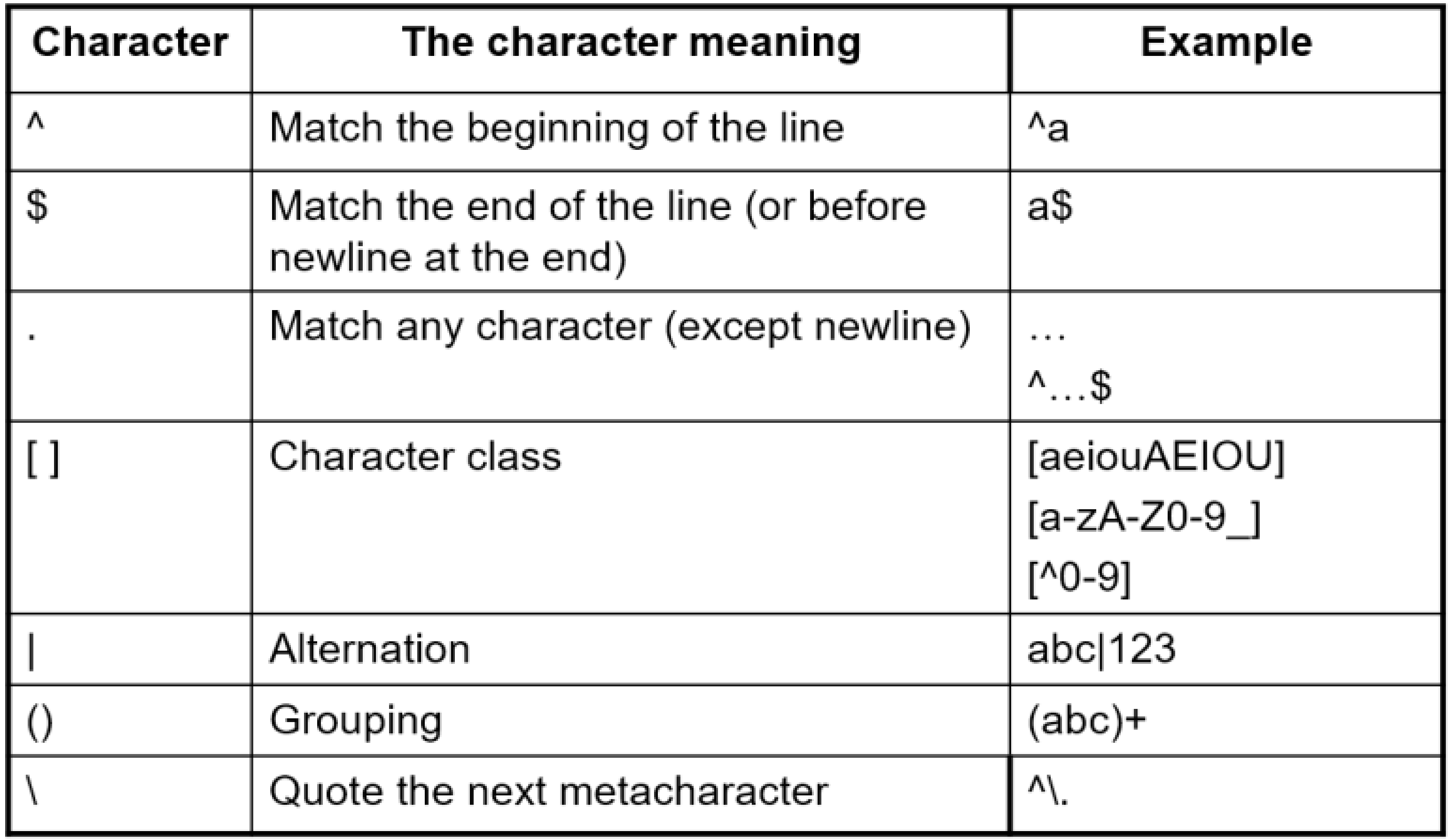
- Concepts
- RE characters
- Search
- Matching Object
- Sub
- Split
- Finditer
- Flags

Concepts About Regular Expressions

- A *regular expression* is a pattern - a template - to be matched against a string.
- Matching a regular expression against a string either succeeds or fails.
- Sometimes, the success or failure may be all you are concerned about and sometimes we to process or to replace the matched pattern.
- Regular expressions are widely used by many programs and languages
- The module **re** provides full support for regular expressions in Python

Regular- expression characters

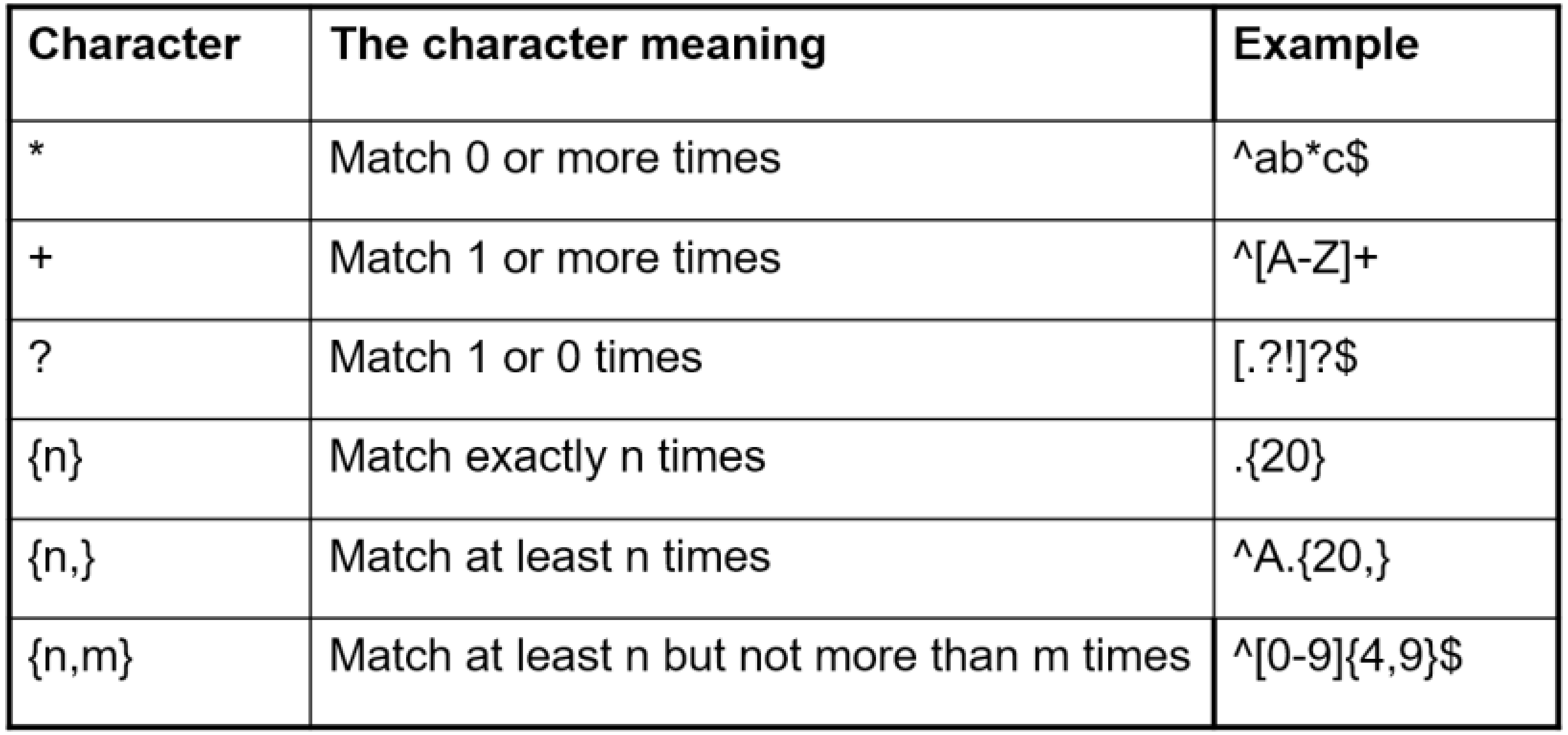
- There is the basic set of regular-expression meaningful characters in Perl.



Character	The character meaning	Example
^	Match the beginning of the line	^a
$	Match the end of the line (or before newline at the end)	a$
.	Match any character (except newline)	… ^…$
[ ]	Character class	[aeiouAEIOU] [a-zA-Z0-9_] [^0-9]
|	Alternation	abc|123
()	Grouping	(abc)+
\	Quote the next metacharacter	^\.

Regular- expression characters - Cont'd

- There is the basic set of quantifiers characters:



Character	The character meaning	Example
*	Match 0 or more times	^ab*c$
+	Match 1 or more times	^[A-Z]+
?	Match 1 or 0 times	[.?!]?$
{n}	Match exactly n times	.{20}
{n,}	Match at least n times	^A.{20,}
{n,m}	Match at least n but not more than m times	^[0-9]{4,9}$

Regular- expression characters - Cont'd

- There is the extended set of Python characters:

Character	**The character meaning**	**Example**
\w	Match a "word" character (alphanumeric plus "_")	^\w{5}$
\W	Match a non-"word" character	^\W.*\W$
\s	Match a whitespace character	\s
\S	Match a non-whitespace character	^\S+$
\d	Match a digit character	\d$
\D	Match a non-digit character	^\D

RE search

- **re.search** - Scan through string looking for the first location where the regular expression pattern produces a match, and return a corresponding MatchObject instance

- match_obj = re.search(pattern, string, flags=0)
    pattern – regular expression
    string – string to look *pattern* into
    flags – possible flags

- match_obj will be None if pattern didn't match

RE search – cont'd

```
import re

line = "my age is 22"
m = re.search(r'(\d+).*', line)
if m:
    print("matched string is {} in index ({},{})".format(
        m.group(1), m.start(1), m.end(1)))
else:
    print("No match!!")
```

RE search – cont'd

```
import re

line = "27:11:2004"
m = re.search(r'(\d+):(\d+):(\d+)', line)
if m:
    print("matched day is {} in index ({},{})".format(
        m.group(1), m.start(1), m.end(1)))
    print("matched month is {} in index ({},{})".format(
        m.group(2), m.start(2), m.end(2)))
    print("matched year is {} in index ({},{})".format(
        m.group(3), m.start(3), m.end(3)))
else:
    print("No match!!")
```

RE sub

- **re.sub** – replaces all (or max) occurrences of the pattern in string. This method would return modified string

- **re.sub(pattern, repl, string, max=0)**
  pattern – regular expression
  repl – replacement string
  string – string to look *pattern* into
  max – maximum replacements

RE sub – cont’d

```python
import re
phone = "2004-959-559"

 # Remove anything other than digits
new_phone = re.sub(r'\D', "", phone)
print ("Phone num now is : ", new_phone )
#Phone num. now is : 2004959559

# Replace '-' with space
new_phone = re.sub(r'-', " ", phone)
print ("Phone num now is : ", new_phone)
#Phone num now is :  2004 959 559
```

RE split

- **re.split** - Split string by the occurrences of pattern

- **re.split(pattern, string, maxsplit=0, flags=0)**
  pattern – regular expression
  string – string to look *pattern* into
  maxsplit – maximum splits
  flags – possible flags

RE split – cont'd

```python
import re
value ="one is 1, two is 2"
result = re.split("[, ]+", value)

for element in result:
 print(element)
```

The Output:

```
one
is
1
two
is
2
```

RE split – cont'd

```
value = "one 1 two 22 three 3"
result = re.split("\D+", value)

for element in result:
 print(element)
```

The Output:

```
1
22
3
```

RE finditer

- **re.finditer** - Return an MatchObject iterator for all matched patterns in string

- **re.finditer(pattern, string, flags=0)**
  pattern – regular expression
  string – string to look *pattern* into
  flags – possible flags

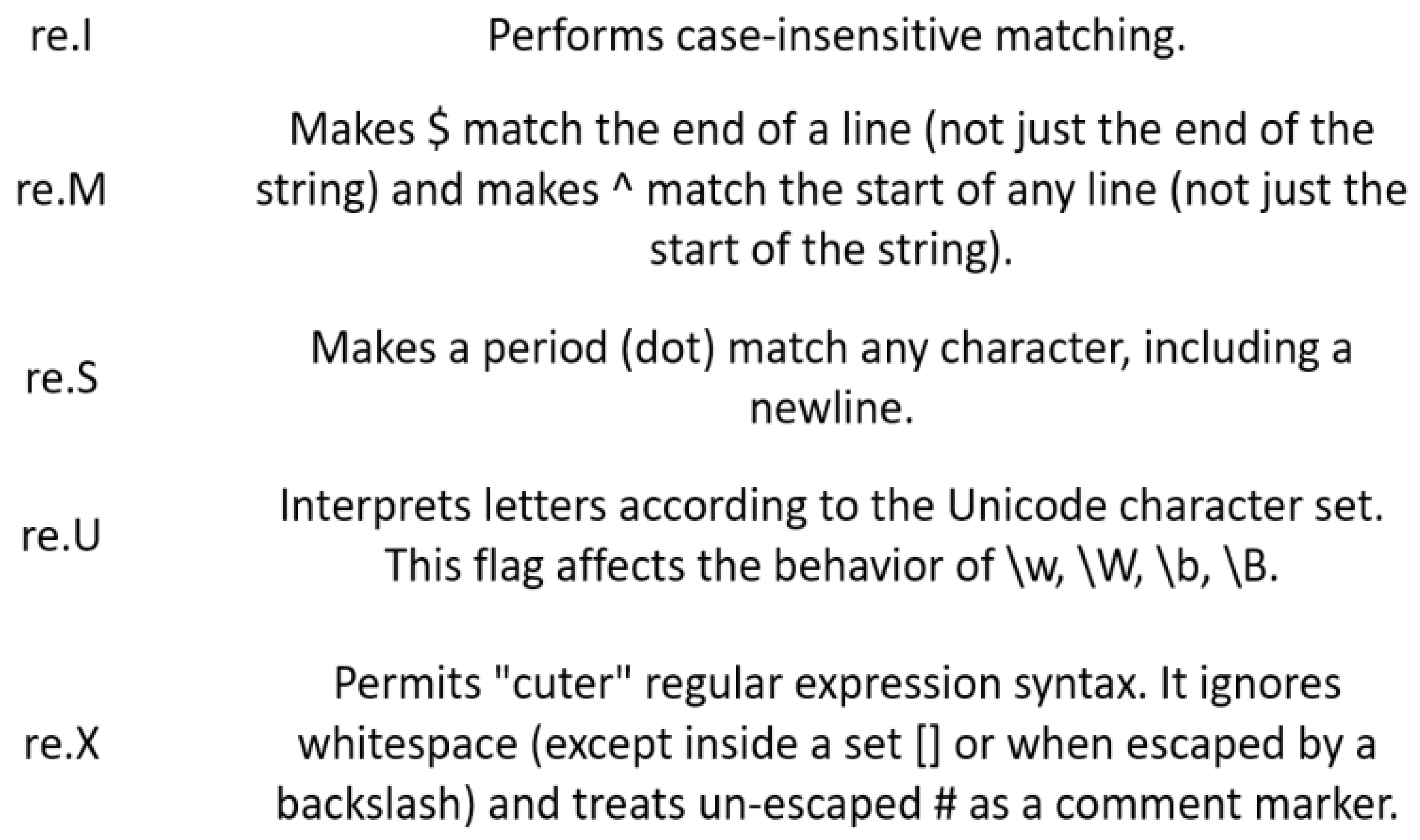
RE finditer – cont'd

```python
import re

text = "this is a long sentence with a lot of words"
for m in re.finditer(r"(\w+)", text):
    print('{}-{}: {}'.format(m.start(1), m.end(1), m.group(1)))
```

```
 0- 4: this
 5- 7: is
 8- 9: a
….
35-37: of
38-43: words
```

RE flags



re.I	Performs case-insensitive matching.
re.M	Makes $ match the end of a line (not just the end of the string) and makes ^ match the start of any line (not just the start of the string).
re.S	Makes a period (dot) match any character, including a newline.
re.U	Interprets letters according to the Unicode character set. This flag affects the behavior of \w, \W, \b, \B.
re.X	Permits "cuter" regular expression syntax. It ignores whitespace (except inside a set [] or when escaped by a backslash) and treats un-escaped # as a comment marker.

Nesting Loops

Demo

Labs 11-12

Lab

Questions