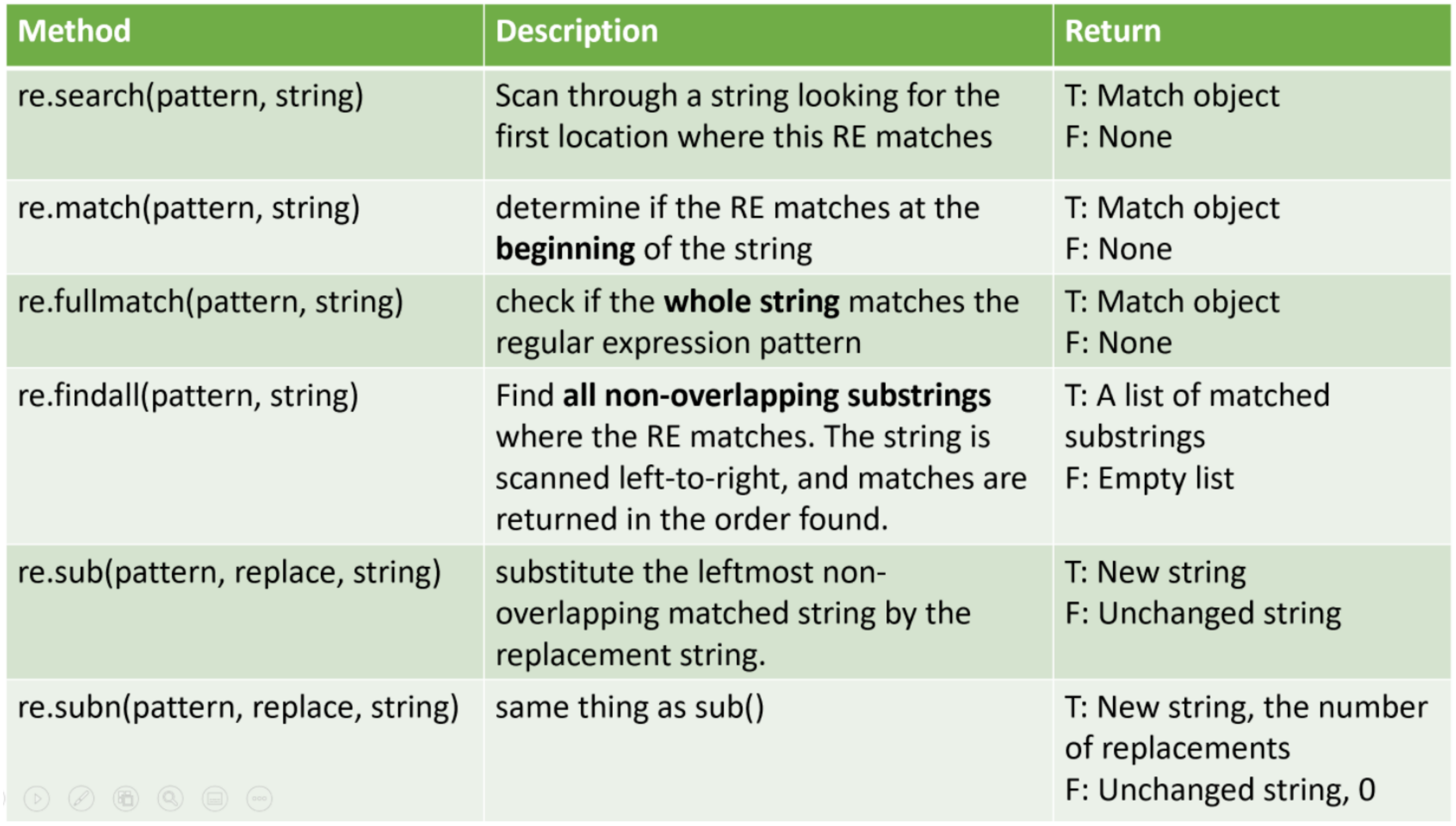
Regular Expression

2021.02.11

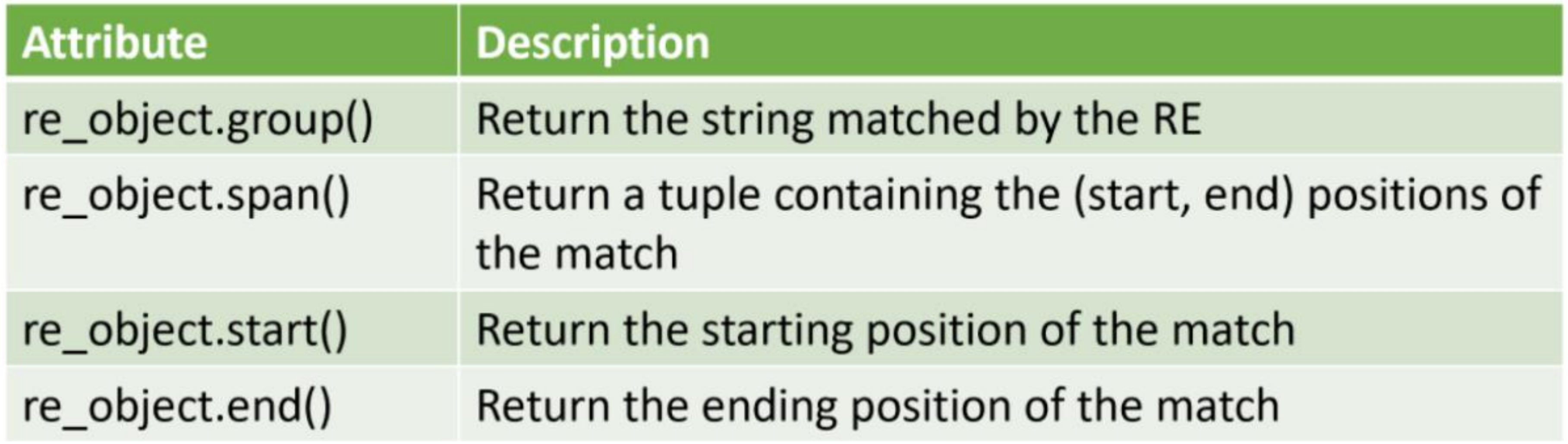
Pattern Matching methods: RE functions



Method	Description	Return
re.search(pattern, string)	Scan through a string looking for the first location where this RE matches	T: Match object F: None
re.match(pattern, string)	determine if the RE matches at the **beginning** of the string	T: Match object F: None
re.fullmatch(pattern, string)	check if the **whole string** matches the regular expression pattern	T: Match object F: None
re.findall(pattern, string)	Find **all non-overlapping substrings** where the RE matches. The string is scanned left-to-right, and matches are returned in the order found.	T: A list of matched substrings F: Empty list
re.sub(pattern, replace, string)	substitute the leftmost non-overlapping matched string by the replacement string.	T: New string F: Unchanged string
re.subn(pattern, replace, string)	same thing as sub()	T: New string, the number of replacements F: Unchanged string, 0

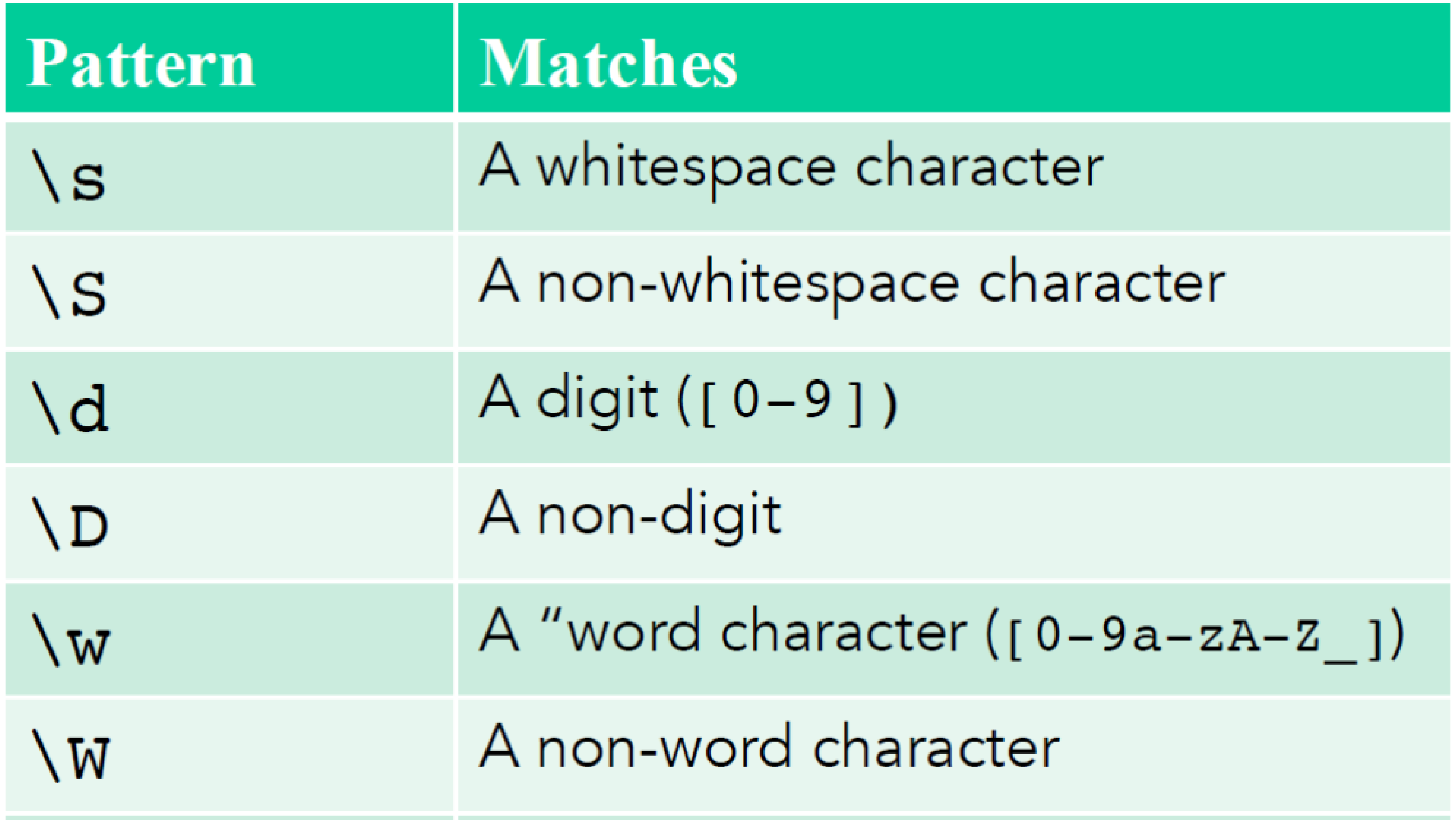
Attributes of Regular Expression objects

RE objects: returned by pattern matching methods



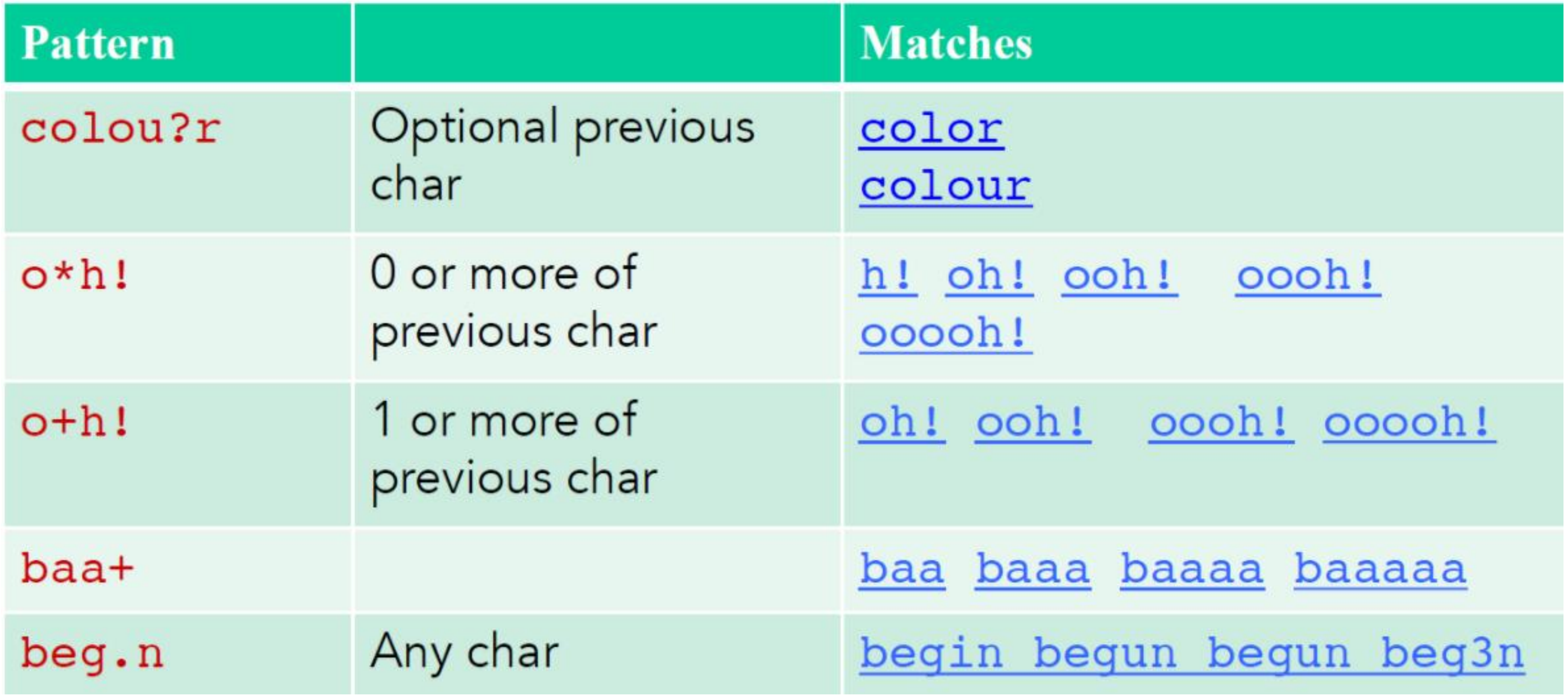
Attribute	Description
re_object.group()	Return the string matched by the RE
re_object.span()	Return a tuple containing the (start, end) positions of the match
re_object.start()	Return the starting position of the match
re_object.end()	Return the ending position of the match

Pre-defined character set



Pattern	Matches
`\s`	A whitespace character
`\S`	A non-whitespace character
`\d`	A digit (`[0-9]`)
`\D`	A non-digit
`\w`	A “word character (`[0-9a-zA-Z_]`)
`\W`	A non-word character

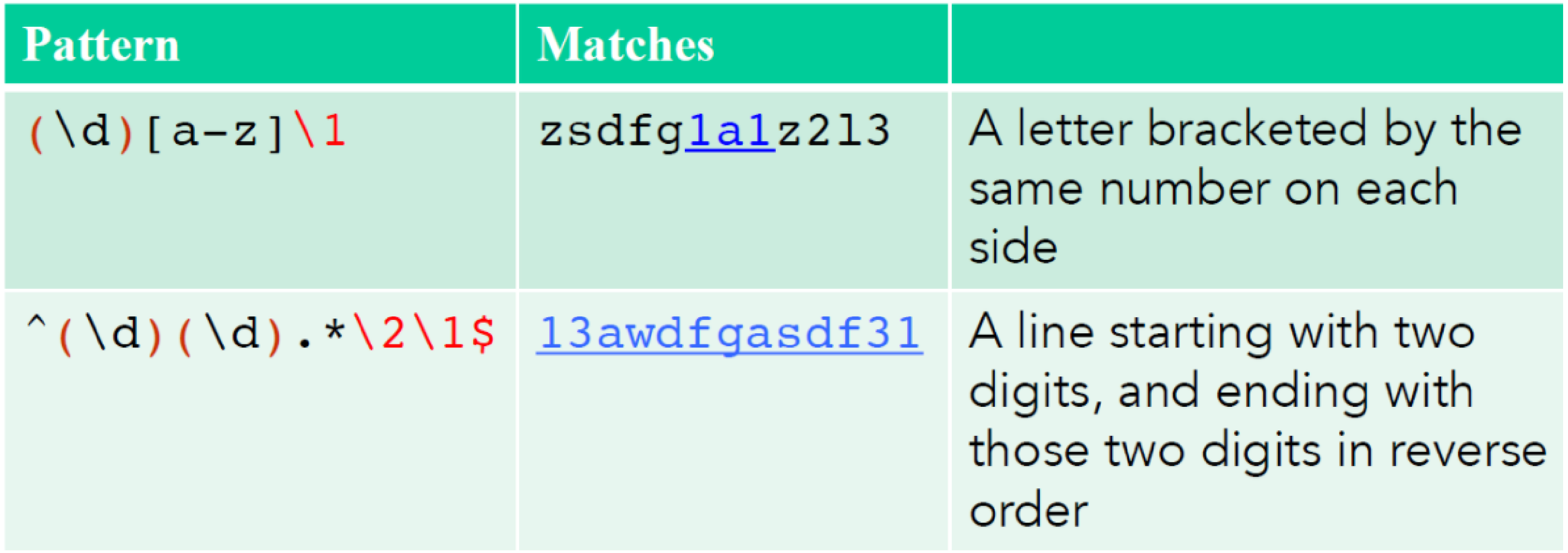
Quantifiers: ? * + .



Pattern		Matches
colou?r	Optional previous char	color colour
o*h!	0 or more of previous char	h! oh! ooh! oooh! ooooh!
o+h!	1 or more of previous char	oh! ooh! oooh! ooooh!
baa+		baa baaa baaaa baaaaa
beg.n	Any char	begin begun begun beg3n

Quantifiers: ( ) and {m,n}

- ( ): capture and group the letters that matched the pattern
- {m,n}: specify the number of repeats of the previous pattern



Pattern	Matches	
`(\d)[a-z]\1`	zsdfg1a1z213	A letter bracketed by the same number on each side
`^(\d)(\d).*\2\1$`	13awdfgasdf31	A line starting with two digits, and ending with those two digits in reverse order

E.g., **a(bc){2,5}** matches a string that has “a” followed by 2 up to 5 repeated sequence “bc”

Anchors: ^ $

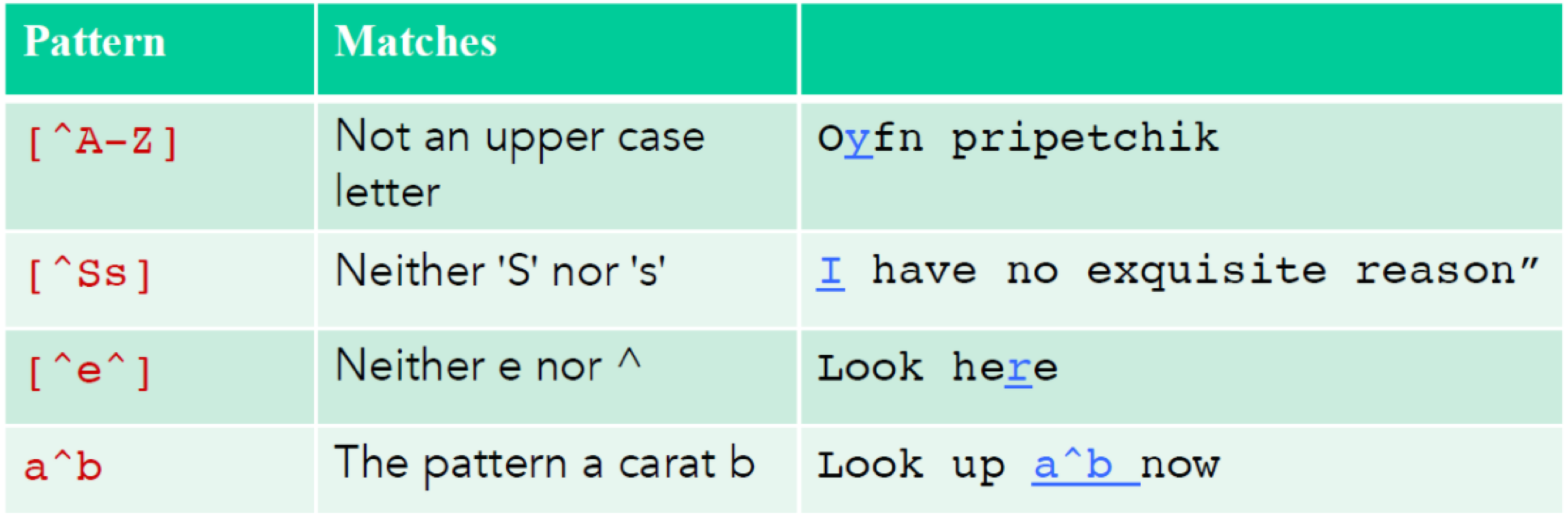
Pattern	Matches	
`^[A-Z]`	`Palo Alto`	Start of string

`^[^A-Za-z]`	`1`
	`“Hello”`	

`\.$`	`The end.`	End of string

`.$`	`The end?`
	`The end!`	

The use cases of ^



Pattern	Matches	
[^A-Z]	Not an upper case letter	Oyfn pripetchik
[^Ss]	Neither 'S' nor 's'	I have no exquisite reason”
[^e^]	Neither e nor ^	Look here
a^b	The pattern a carat b	Look up a^b now

Pattern	Description
[^....]	negation: the matched string don't contain any character inside the square bracket
[..^..]	matches the actual ^ character
r'..\^..'	matches the actual ^ character
r'^a....'	matches a string that starts with 'a'

OR operator: pipe |

A choice between / among elements separated by pipe

Pattern	Matches
groundhog|woodchuck	groundhog woodchuck
yours|mine	yours mine
a|b|c	= [abc]
[gG]roundhog|[Ww]oodchuck	...

OR operator: square bracket [ ]

Match one of the letter inside the square bracket

Pattern	Matches
`[wW]oodchuck`	Woodchuck, woodchuck
`[1234567890]`	Any digit

Pattern	Matches	
`[A-Z]`	An upper case letter	`Drenched Blossoms`
`[a-z]`	A lower case letter	`my beans were impatient`
`[0-9]`	A single digit	`Chapter 1: Down the Rabbit Hole`

Backslash: \

Backslash as escape:

- deprive the special power of the character
- avoid confusion for characters that have special meaning
- literally match a specific character

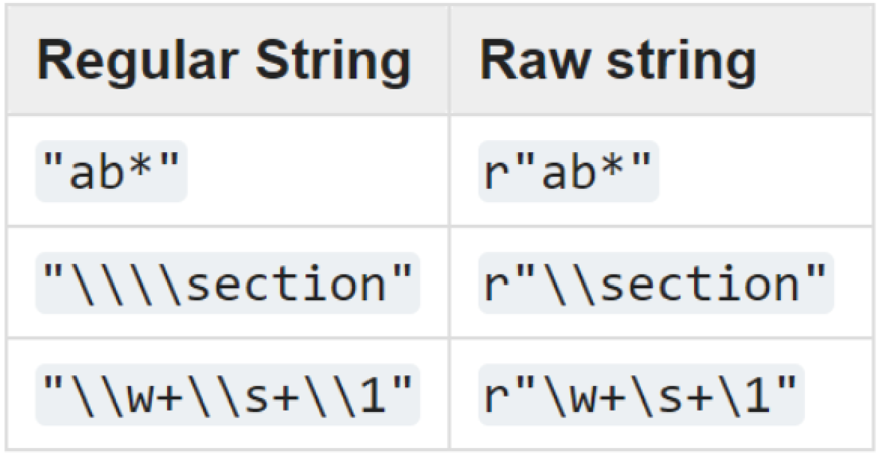
Characters	Stage
`\section`	Text string to be matched
`\\section`	Escaped backslash for `re.compile()`
`"\\\\section"`	Escaped backslashes for a string literal

```
re.search(r'\.', 'The end.') # escape
```

```
<re.Match object; span=(7, 8), match='.'>
```

r

raw string notation for regular expression patterns in Python



Regular String	Raw string
`"ab*"`	`r"ab*"`
`"\\\\section"`	`r"\\section"`
`"\\w+\\s+\\1"`	`r"\w+\s+\1"`

Examples

(1) Extract the phone number in the following text:

text = "Call 414-555-1212 for info“

(2) Extract the email address in the following text:

text = “Please contact zwang185@hawk.iit.edu for detailed info”

(3) Find and correct the mis-spellings (e.g., pythn -> python)

text = “We are using python for CS481;

Do you use pythn for your class?

What’s the advantages of using pythn?”