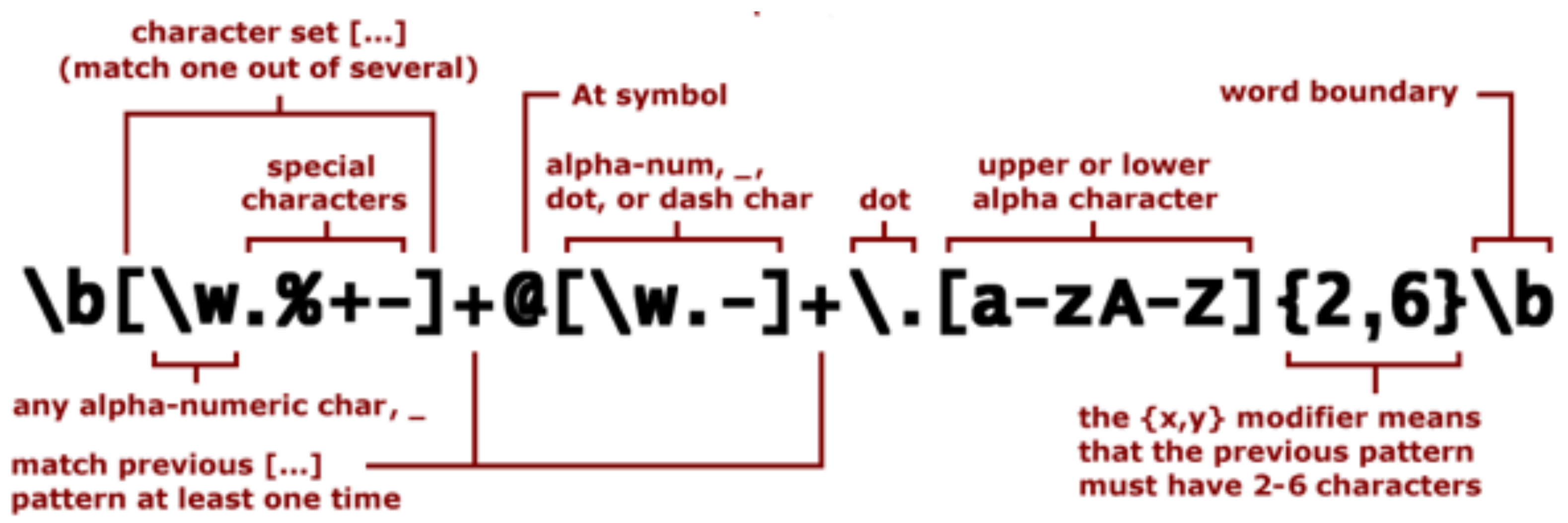
Python Regular Expression

Python Essentials 2015

Benny Khoo (17 Oct 2015)


character set [...]
(match one out of several)
special characters
At symbol
alpha-num, _, dot, or dash char
dot
upper or lower alpha character
word boundary
\b[\w.%+-]+@[\w.-]+\.[a-zA-Z]{2,6}\b
any alpha-numeric char, _
match previous [...] pattern at least one time
the {x,y} modifier means that the previous pattern must have 2-6 characters

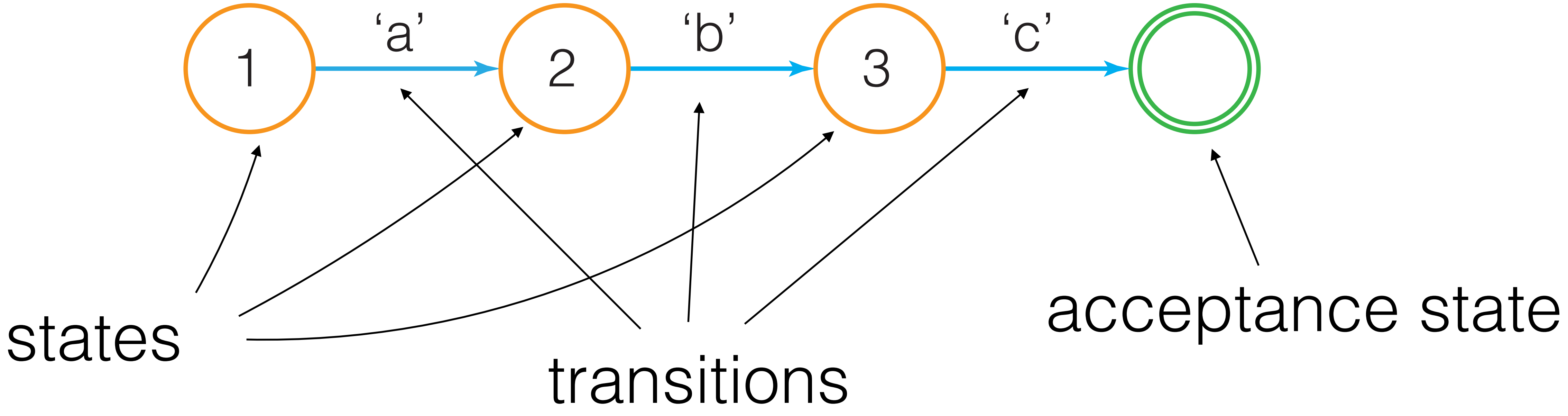
Introducing RE

```python
import re
```

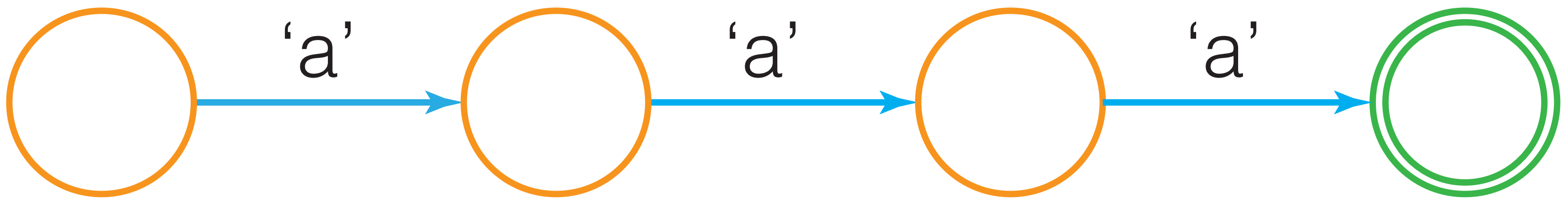
Thinking about RE

```python
import re

line = 'abcdef'
re.match('abc', line)
```



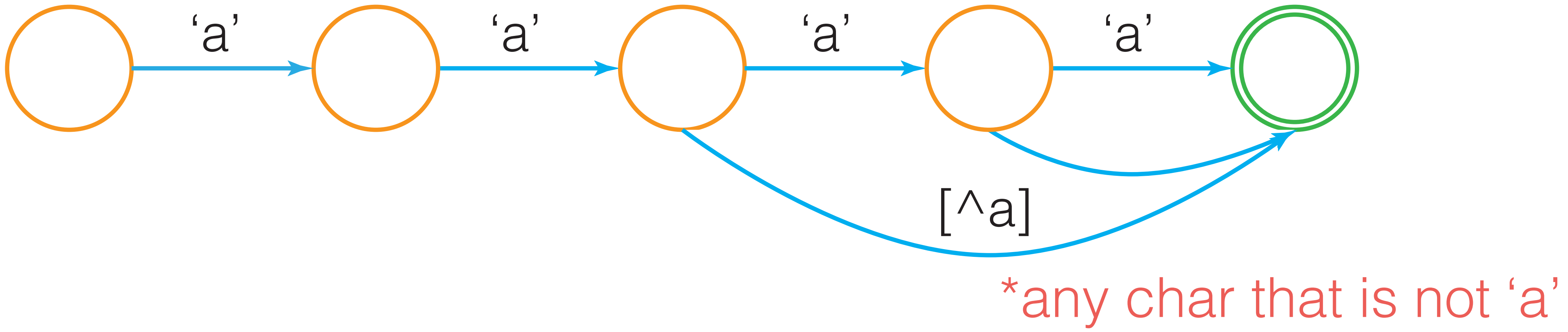
a{*n*}



a{3}

- Match a character **exactly** *n* times.

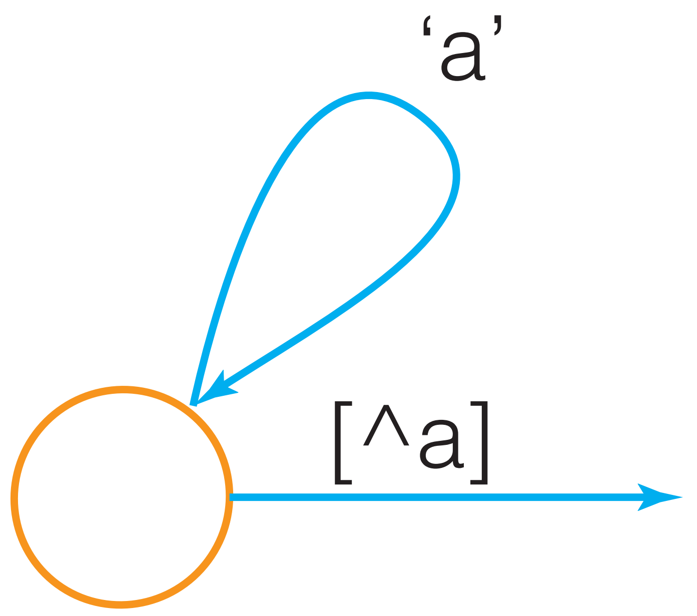
a{*m*,*n*}



a{2,4}

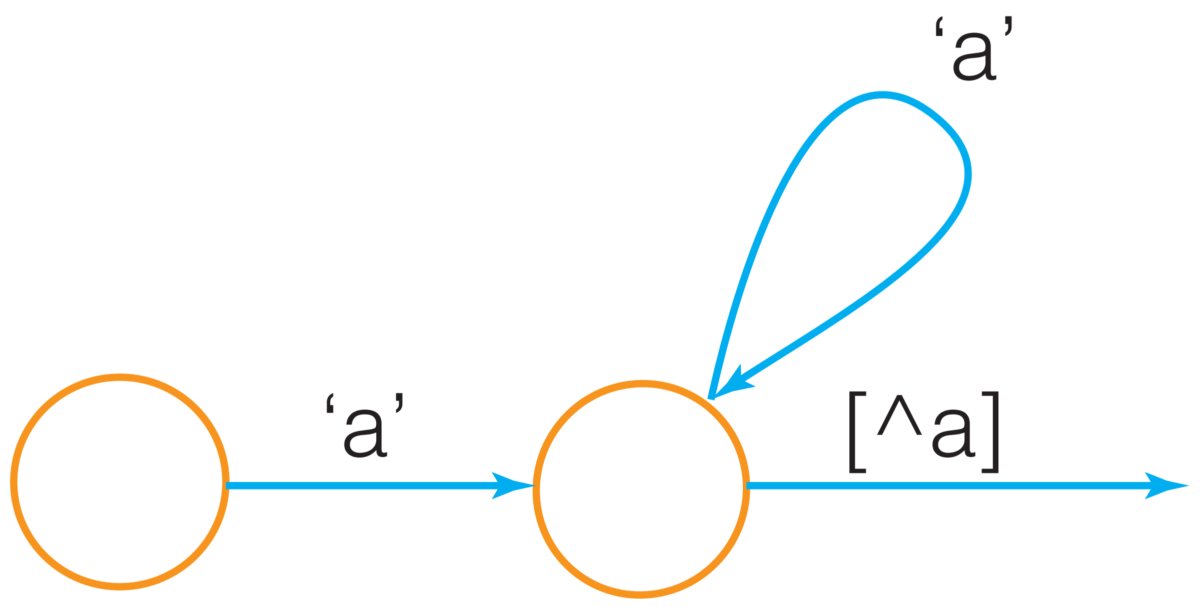
- Match at least *m* to maximum *n* times.

a*



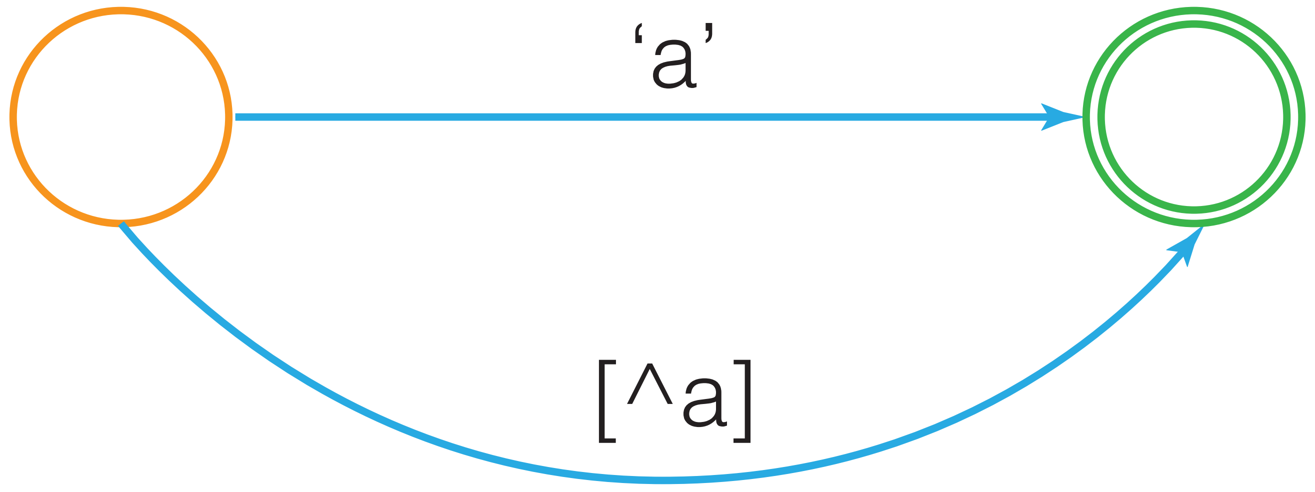
- Match zero or multiple times.

a+



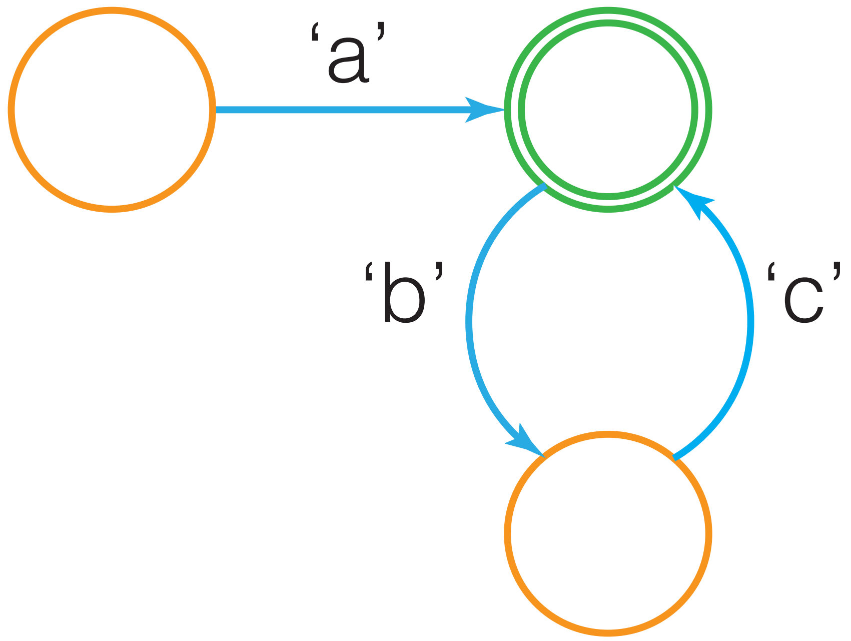
- Match at least one or multiple times.

a?



- Match zero or one time.

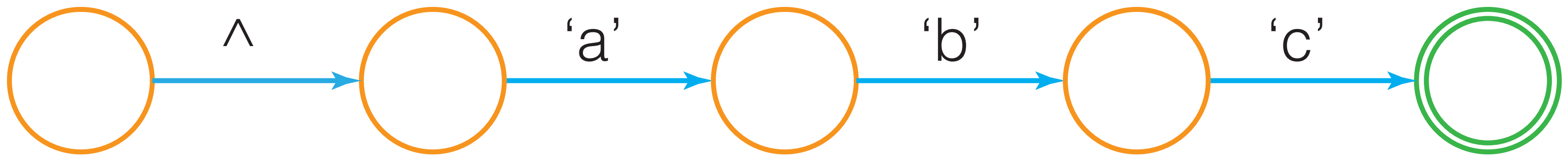
a(bc)*



✓ a
✓ abcbc
✓ abcbcbc
✗ abcb

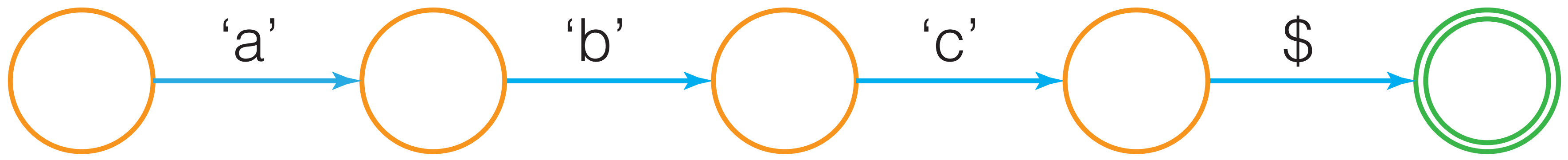
- Match a grouped sequence zero or multiple times

Starting Anchor: ^abc



- ‘^’ match the starting position of a input sequence

End Anchor: abc$



- ‘$’ match the termination of a input sequence

A range of character [A-Z]

[A-Za-z]	match a single alphabet
[0-9]	match a single digit
[\n\r\t\f]	match a whitespace character
[_A-Za-z0-9]	match a single alphanumeric character
[A-Za-z0-9_.@]	match a valid character set in email address?

Negated character set

```
[^A-Za-z_]

[^ ]

“[^”]+”

<[^>]+>
```

Character set shorthand

\d	Matches any decimal digit; this is equivalent to [0-9]. and unicode variants.
\D	Opposite of \d
\s	Matches characters considered whitespace in the ASCII character set; this is equivalent to [\t\n\r\f\v].
\S	Opposite of \s
\w	Matches characters considered alphanumeric in the ASCII character set; this is equivalent to [a-zA-Z0-9_].
\W	Opposite of \w
\b	Matches the empty string, but only at the beginning or end of a word.
\B	Opposite of \b

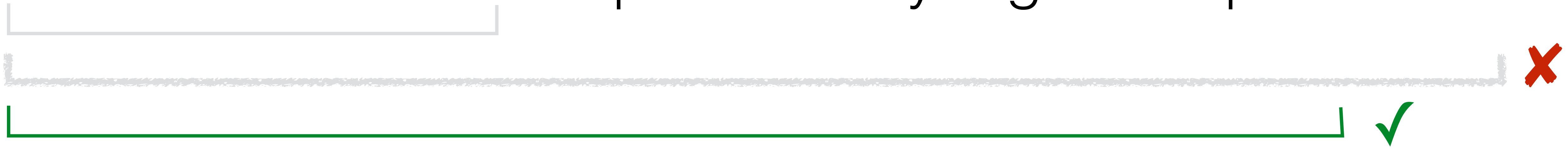
Wild card

.	Match **any** character
.?	Match any zero or single character
.*	Match zero or as many character
.+	Match at least one or as many character

Greedy regex

.*es

Essential services are provided by regular expressions.



Regex by default produces match with the longest span

The engine knows to discard inessential matches to fulfil the larger goal.

Match minimally

.*?es

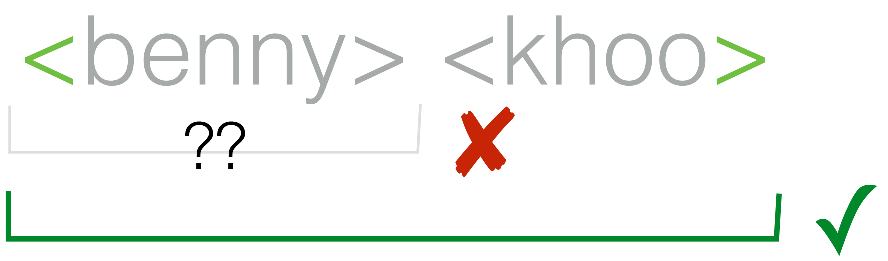
Essential services are provided by regular expressions.

Shortest span that still match the regex completely

Greedy regex 2

```
<.+>
```

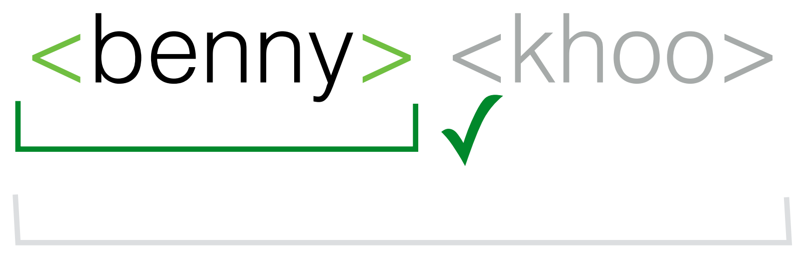
```
<[^>]+>
```



Minimal version

```
<.+?>

<[^>]+>
```



RE functions

- Test or match a given string to pattern: re.match, re.fullmatch
- Search the string for the first presence of pattern: re.search
- Find and extract all occurrence of pattern in string: re.findall, re.finditer
- Making modification to string on presence of pattern: re.sub, re.subn, re.split
- Compiling a regex for reuse: re.compile

Search vs. Match

re.match	re.search
Match *start* at beginning in the string.	Match the *first* occurrence of pattern *anyway* in the string.
re.match(“abc”)	is synonymous to re.search(“^abc”)
re.fullmatch(“abc”)	is synonymous to re.search(“^abc$”)
Useful idiom to validate user input string against some expecting pattern e.g. date formatting, valid phone number.	Search a block of text for data.

Match pattern multiply

- re.search stops at the first occurrence of pattern.It does not move further from the first hit.

??

```
line = '< ... < ... > ... >'
re.search(r'<', line)
# <_sre.SRE_Match object; span=(0, 1), match='<'>
```

- To be able to search further you probably need to bump up the start index.

```
re.search(r'<', line[m.end():])
<_sre.SRE_Match object; span=(4, 5), match='<'>
```

re.findall

- Search for multiple occurrences of a pattern in a file.

```
re.findall(r'".*?"', f.read())
['"http://www.lazada.com.my/hisense-32-led-hd-tv-hmled32d33-539003.html"',
 '"HI091ELALJWBANMY-572873"',
 ...
```

- re.findall returns a tuple when regex contains capture patterns.

```
re.findall(r'(\w+) (\w+)', line)
[('quick', 'brown'), ('fox', 'jumps'), ('over', 'the'), ('lazy', 'dog')]
```

- re.findall does not return a match object. In some case it is essential. Use re.finditer instead.

RE options

- Most re functions accept bitwise OR flags option in the last argument.

re.I re.IGNORECASE	Perform case-insensitive matching; expressions like [A-Z] will match lowercase letters, too.
re.M re.MULTILINE	Match multiple line in input stream. This changes the behaviour of ^ and $.

re.X
re.VERBOSE
	Ignore whitespaces in regex.
	`a = re.compile(r"""\d +  # the integral part`
	`                   \.    # the decimal point`
	`                   \d *  # some fractional digits""", re.X)`
	`# or rather`
	`b = re.compile(r"\d+\.\d*")`

Raw string notation

- To keep you sane from multiple escape always stick to the raw string notation.

```
>>> re.match(r"\W(.)\1\W", " ff ")
<_sre.SRE_Match object; span=(0, 4), match=' ff '>
>>> re.match("\\W(.)\\1\\W", " ff ")
<_sre.SRE_Match object; span=(0, 4), match=' ff '>

>>> re.match(r"\\", r"\\")
<_sre.SRE_Match object; span=(0, 1), match='\\'>
>>> re.match("\\\\", r"\\")
<_sre.SRE_Match object; span=(0, 1), match='\\'>
```

Capturing sub-match

```
>>> m = re.match(r"(\w+) (\w+)", "Isaac
Newton, physicist")

>>> m
<_sre.SRE_Match object; span=(0, 12),
match='Isaac Newton'>

>>> m.group(0)
'Isaac Newton'

>>> m.group(1)
'Isaac'

>>> m.group(2)
'Newton'


>>> m = re.match(r"(?P<first_name>\w+) (?P<last_name>\w+)", "Malcolm Reynolds")
>>> m.group('first_name')
'Malcolm'
>>> m.group('last_name')
‘Reynolds'

>>> m.group(1)
'Malcolm'
>>> m.group(2)
'Reynolds'
```

Match objects

m.**group**([group1, ...])	Returns one or more subgroups of the match.
m.**start**([group]) *m*.**end**([group])	Return the indices of the start and end of the substring matched by group; group defaults to zero (meaning the whole matched substring).
m.**span**([group])	For a match m, return the 2-tuple (m.start(group), m.end(group)).

- Most *re* functions return a match object except *findall*.

Pattern objects

```
for line in file:
    re.match(r’<\s*(/?\w+).*?>', line)
```

compare to

```
tagPattern = re.compile(r’<\s*(/?\w+).*?>')
for line in file:
    tagPattern.match(line) # or re.match(tagPattern, line)
```

- Regex can be slow due to compilation cycle especially when it has to be accessed numerous times in a loop.
- Most *re* functions seen are accessible as OO interface too e.g. a *p* object: *p*.search, *p*.match, *p*.findall, *p*.finditer, *p*.sub, *p*.split so on.

Sample regex

IPv4	(\d{1,3}+.){3}\d{1,3}+	1.2.3.4 192.168.2.1
Number between 0-255	(?:25[0-5]) # 250-255 |(?:2[0-4][0-9]) # 200-249 |(?:1[0-9]{2}) # 100-199 |(?:[0-9]{1,2}) # 0-99	
C identifier	[_a-zA-Z][_a-zA-Z0-9]*	var1 var2
Quoted string	"([^"](\\.)?)*"	“benny khoo” “benny \”khoo\””
Pair of same word	(\w+)-\1	sama-sama main-main

Conditional regex

(?=…)	Matches preceding expression if … matches.	Match any name with the surname “Khoo”. r'\w+ (?=Khoo)'
(?!…)	Matches preceding expression if … does not match.	
(?<=…)	Matches following expression if … matches	Match any name with first name “Benny”. r'(?<=Benny) \w+’
(?<!…)	Matches following expression if … does not matches	

Conditional regex 2

```
(?(id/name)yes-pattern|no-pattern)
```

- Match *yes-pattern* if id/name presence otherwise match *no-pattern*.

- Match a quoted string or a variable name: -

```
((?P<quote>”)[^"]+"|\S+)
```

Substitution

Substitute a pattern with capture group

```
>>> re.sub('(\w+)', r'_\1', 'prefix with underscore')
'_prefix _with _underscore'
```

Substitute a pattern with function

```
>>> def power(m):
    return "{}^{} = {}".format(m.group(1), m.group(2),
int(m.group(1))**int(m.group(2)))

>>> re.sub(r'(\d+)\^(\d+)', power, 'sub is powerful 3^2')
'sub is powerful 3^2 = 9'
```

Splitting

Split a string by delimited character

```
>>> re.split(r'\s*,\s*', 'comma, separated, values, 123.123')
['comma', 'separated', 'values', '123.123']
```

Splitting with negated set pattern

```
>>> re.split(r'\W+', 'comma, separated, values, 123.123')
['comma', 'separated', 'values', '123', '123']
```

Problems

- Replace text with the prefix $*nn.nn* (dollar values) with equivalent currency value in Malaysia ringgit. For instance: -

In the case of the new S6 Edge, the 64GB version sells for US$799.99 in the United States (without a carrier subsidy), less than the retail price of US$849.99 for the iPhone 6 Plus with the same storage capacity.

In the case of the new S6 Edge, the 64GB version sells for US$799.99 (RM2,902) in the United States (without a carrier subsidy), less than the retail price of US$849.99 (RM3,083) for the iPhone 6 Plus with the same storage capacity.

Problems

- Find the company name in the following news article with the stock quote info of the day.

> Many industry experts believe that Intel Corporation might be the one that got its hands on the EUV (extreme ultraviolet) technology. ASML shares jumped 12% to around $109 in early trading today.

A. Assuming in this article there is only two known companies in your database “Intel” and “ASML”

B. Use the following URL query to fetch the quote price.

```
http://download.finance.yahoo.com/d/quotes.csv?s=INTC,ASML&f=nsl1c0c1&e=.csv
```

* doc

Problems

- Web scraping
  - List out all item price in a hot deals page.
  - List out images in a web page.
- String template:
  - https://weblogs.java.net/blog/aberrant/archive/2010/05/25/using-stringtemplate-part-1-introduction-stringtemplate