Regular Expressions

Chapter 11

Python for Informatics: Exploring Information
www.pythonlearn.com

Regular Expressions

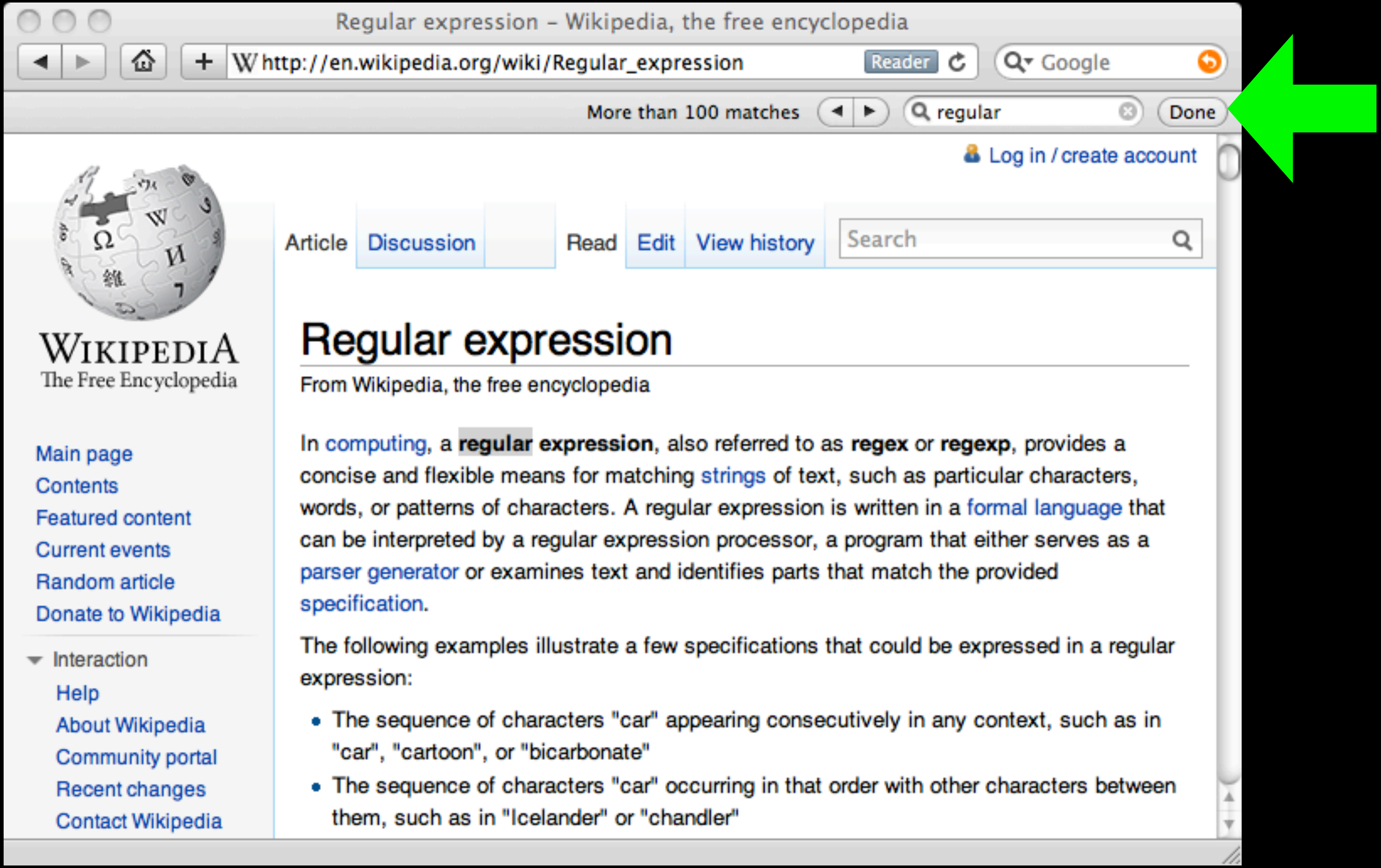
In computing, a regular expression, also referred to as "regex" or "regexp", provides a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters. A regular expression is written in a formal language that can be interpreted by a regular expression processor.

http://en.wikipedia.org/wiki/Regular_expression

Regular Expressions

Really clever "wild card" expressions for matching and parsing strings.

http://en.wikipedia.org/wiki/Regular_expression



Regular expression

From Wikipedia, the free encyclopedia

In computing, a **regular expression**, also referred to as **regex** or **regexp**, provides a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters. A regular expression is written in a formal language that can be interpreted by a regular expression processor, a program that either serves as a parser generator or examines text and identifies parts that match the provided specification.

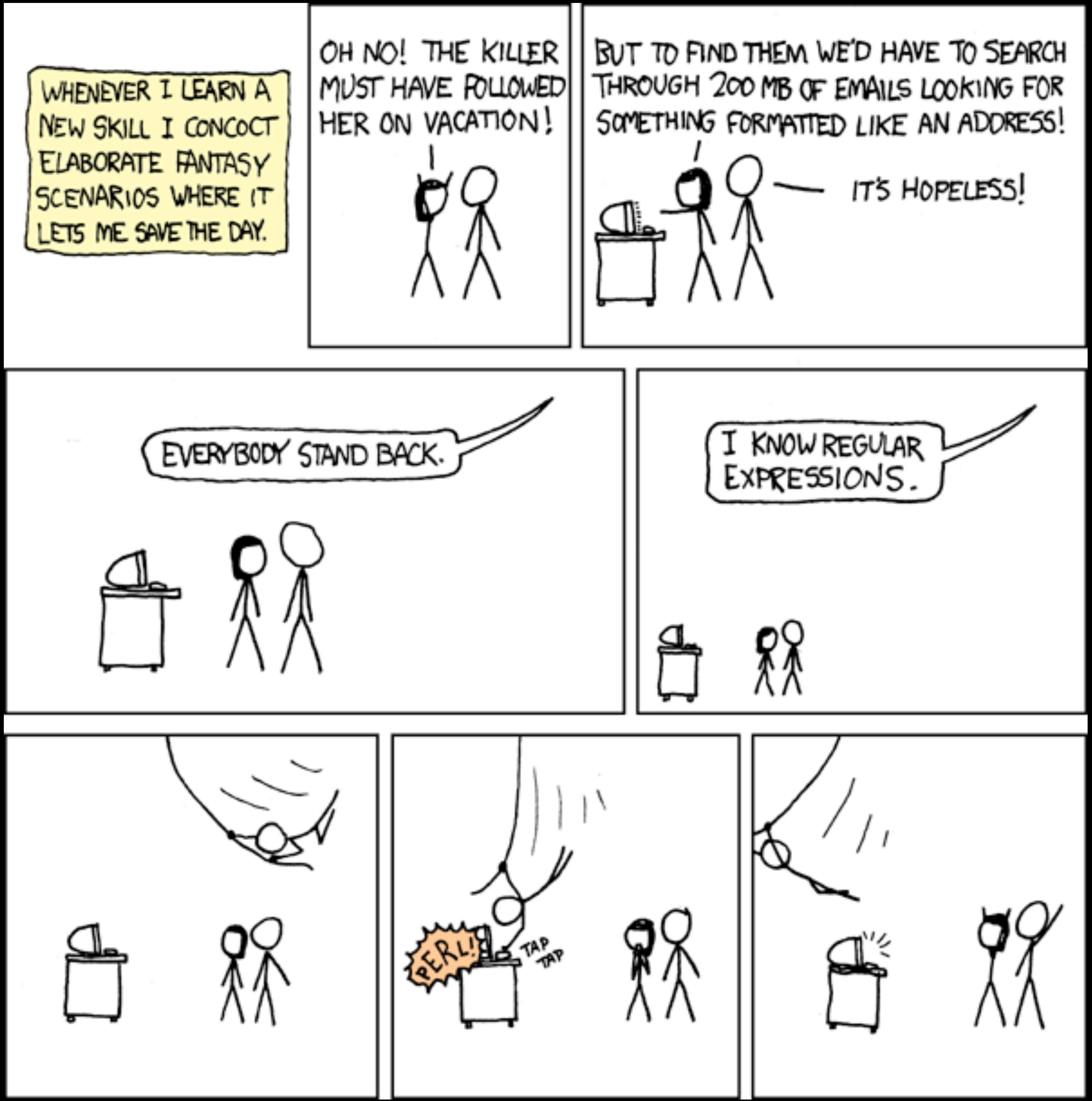
The following examples illustrate a few specifications that could be expressed in a regular expression:

- The sequence of characters "car" appearing consecutively in any context, such as in "car", "cartoon", or "bicarbonate"
- The sequence of characters "car" occurring in that order with other characters between them, such as in "Icelander" or "chandler"

Really smart "Find" or "Search"

Understanding Regular Expressions

- Very powerful and quite cryptic
- Fun once you understand them
- Regular expressions are a language unto themselves
- A language of "marker characters" - programming with characters
- It is kind of an "old school" language - compact


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

http://xkcd.com/208/

Regular Expression Quick Guide

^	Matches the beginning of a line
$	Matches the end of the line
.	Matches any character
\s	Matches whitespace
\S	Matches any non-whitespace character
*	Repeats a character zero or more times
*?	Repeats a character zero or more times (non-greedy)
+	Repeats a chracter one or more times
+?	Repeats a character one or more times (non-greedy)
[aeiou]	Matches a single character in the listed set
[^XYZ]	Matches a single character not in the listed set
[a-z0-9]	The set of characters can include a range
(	Indicates where string extraction is to start
)	Indicates where string extraction is to end

The Regular Expression Module

- Before you can use regular expressions in your program, you must import the library using "import re"
- You can use re.search() to see if a string matches a regular expression similar to using the find() method for strings
- You can use re.findall() extract portions of a string that match your regular expression similar to a combination of find() and slicing: var[5:10]

Using re.search() like find()

```
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if line.find('From:') >= 0:
        print line
```

```
import re

hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('From:', line) :
        print line
```

Using re.search() like startswith()

```
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if line.startswith('From:') :
        print line
```

```
import re

hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:', line) :
        print line
```

We fine-tune what is matched by adding special characters to the string

Wild-Card Characters

- The dot character matches any character
- If you add the asterisk character, the character is "any number of times"

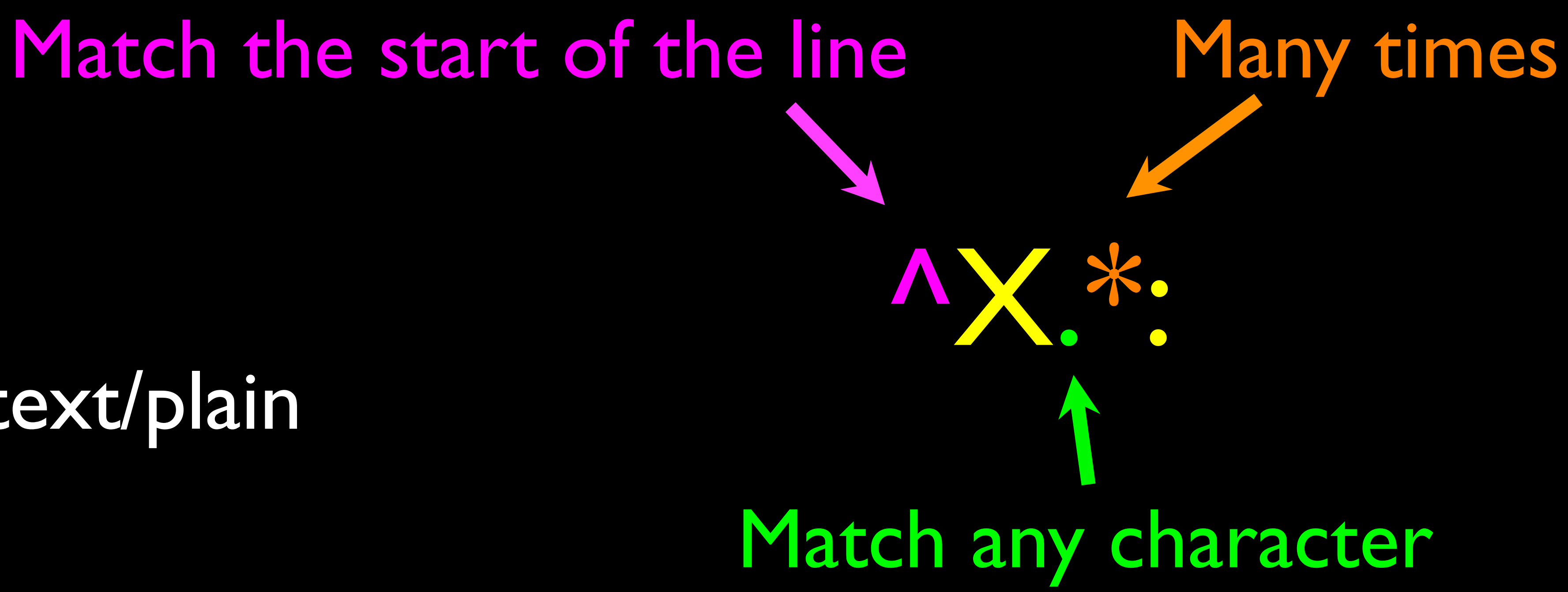
```
X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-DSPAM-Confidence: 0.8475
X-Content-Type-Message-Body: text/plain
```

```
^X.*:
```

Wild-Card Characters

- The dot character matches any character
- If you add the asterisk character, the character is "any number of times"

```
X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-DSPAM-Confidence: 0.8475
X-Content-Type-Message-Body: text/plain
```

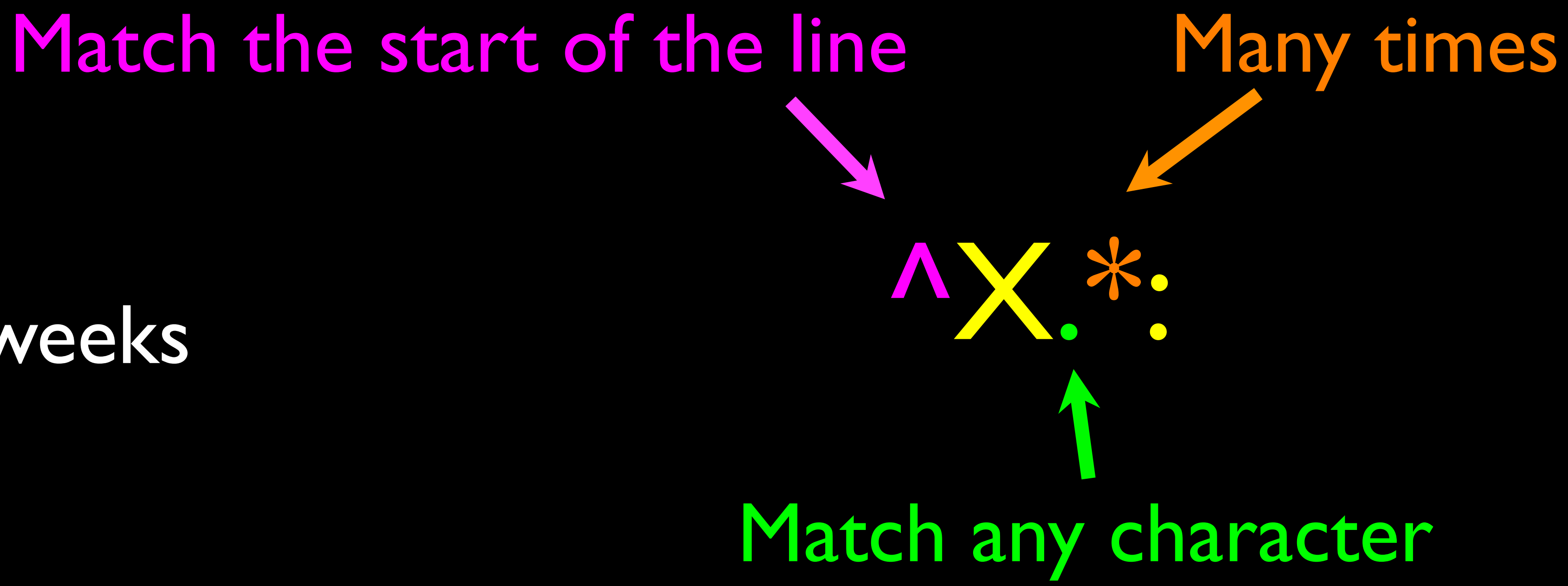


```
^X.*:
```

Fine-Tuning Your Match

- Depending on how "clean" your data is and the purpose of your application, you may want to narrow your match down a bit

```
X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-Plane is behind schedule: two weeks
```



```
^X.*:
```

Fine-Tuning Your Match

- Depending on how "clean" your data is and the purpose of your application, you may want to narrow your match down a bit

X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-Plane is behind schedule: two weeks

Match the start of the line

One or more times

^X-\S+:

Match any non-whitespace character

Matching and Extracting Data

- The re.search() returns a True/False depending on whether the string matches the regular expression
- If we actually want the matching strings to be extracted, we use re.findall()

[0-9]+

↑

One or more digits

```
>>> import re
>>> x = 'My 2 favorite numbers are 19 and 42'
>>> y = re.findall('[0-9]+',x)
>>> print y
['2', '19', '42']
```

Matching and Extracting Data

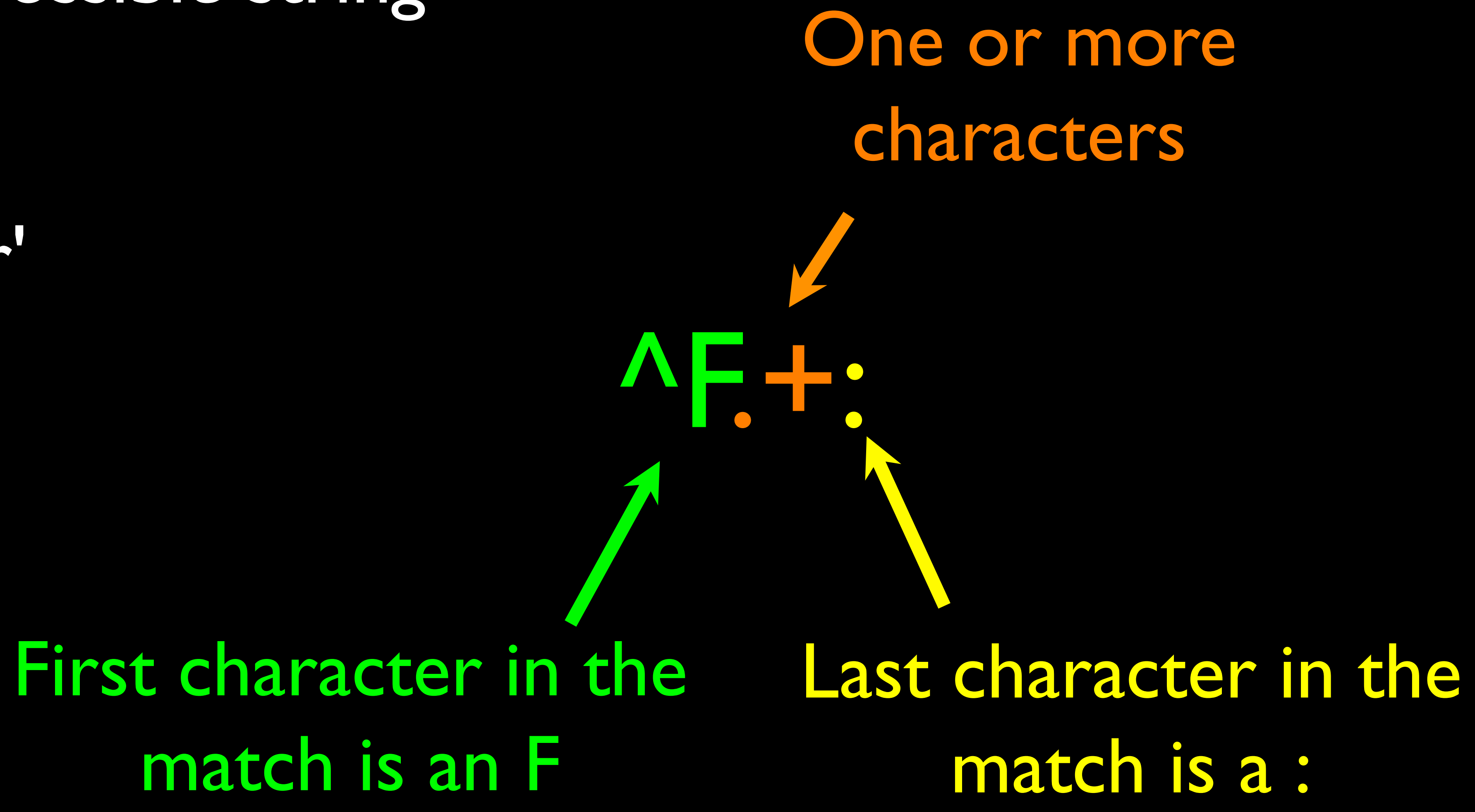
- When we use re.findall() it returns a list of zero or more sub-strings that match the regular expression

```
>>> import re
>>> x = 'My 2 favorite numbers are 19 and 42'>>> y = re.findall('[0
>>> y = re.findall('[AEIOU]+',x)
>>> print y
[]
```

Warning: Greedy Matching

- The repeat characters (* and +) push outward in both directions (greedy) to match the largest possible string

```
>>> import re
>>> x = 'From: Using the : character'
>>> y = re.findall('^F.+:', x)
>>> print y
['From: Using the :']
```

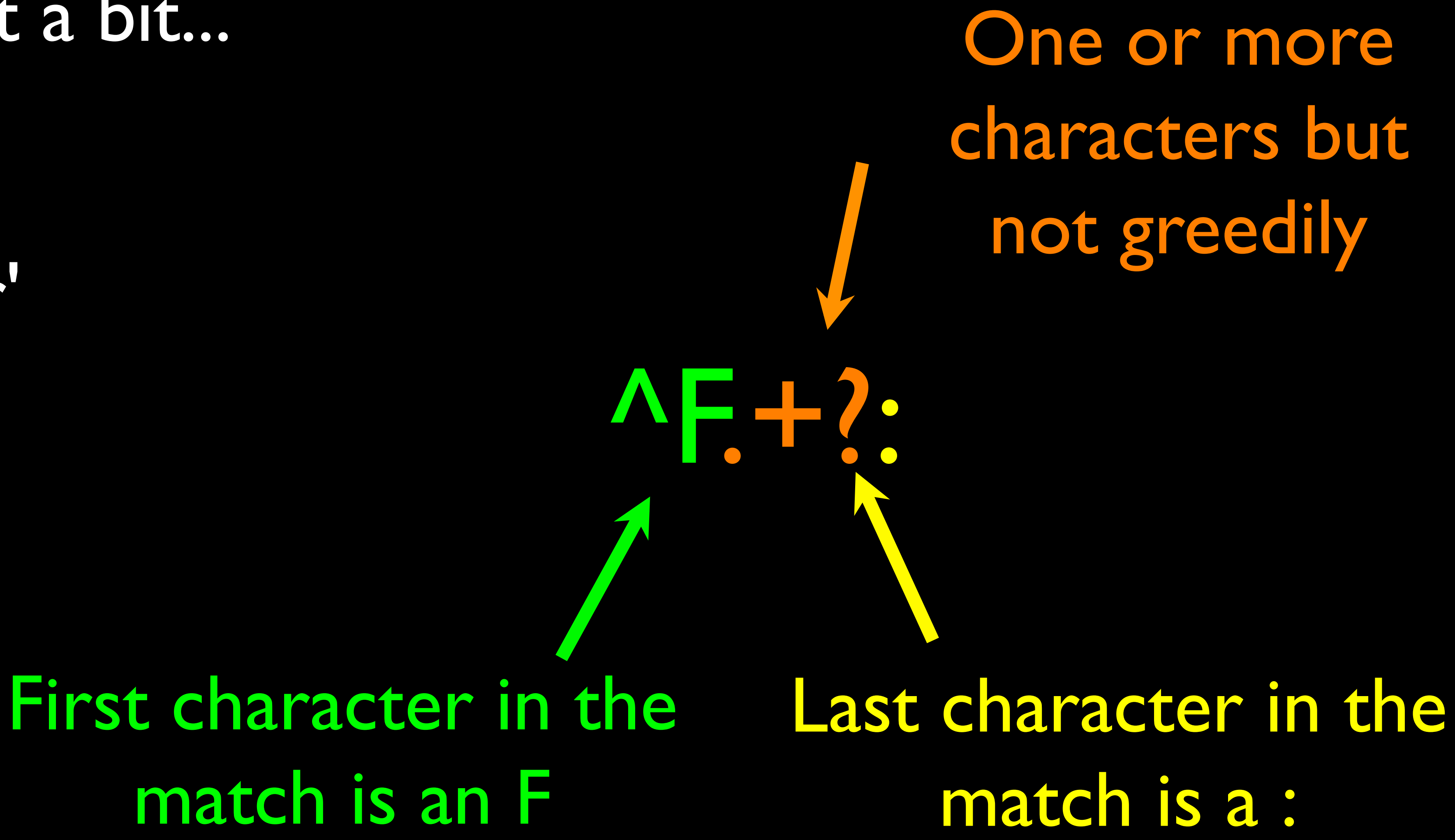

One or more characters
^F.+:
First character in the match is an F
Last character in the match is a :

Why not 'From:'?

Non-Greedy Matching

- Not all regular expression repeat codes are greedy! If you add a ? character - the + and * chill out a bit...

```
>>> import re
>>> x = 'From: Using the : character'
>>> y = re.findall('^F.+?:', x)
>>> print y
['From:']
```


One or more characters but not greedily
^F.+?:
First character in the match is an F
Last character in the match is a :

Fine Tuning String Extraction

- You can refine the match for re.findall() and separately determine which portion of the match that is to be extracted using parenthesis

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
>>> y = re.findall('\S+@\S+',x)
>>> print y
['stephen.marquard@uct.ac.za']
```

`\S+@\S+`

At least one non-whitespace character

Fine Tuning String Extraction

- Parenthesis are not part of the match - but they tell where to start and stop what string to extract

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
>>> y = re.findall('\S+@\S+',x)
>>> print y
['stephen.marquard@uct.ac.za']
>>> y = re.findall('^From (\S+@\S+)',x)
>>> print y
['stephen.marquard@uct.ac.za']
```

$$\texttt{\^{}From } \underset{\uparrow}{(}\texttt{\textbackslash S+@\textbackslash S+}\underset{\uparrow}{)}$$

```
                     21        31
                     ↓         ↓
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print atpos
21
>>> sppos = data.find(' ',atpos)
>>> print sppos
31
>>> host = data[atpos+1 : sppos]
>>> print host
uct.ac.za
```

Extracting a host name - using find and string slicing.

The Double Split Version

- Sometimes we split a line one way and then grab one of the pieces of the line and split that piece again

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

The Double Split Version

- Sometimes we split a line one way and then grab one of the pieces of the line and split that piece again

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
words = line.split()
email = words[1]
pieces = email.split('@')
print pieces[1]
```

```
stephen.marquard@uct.ac.za

['stephen.marquard', 'uct.ac.za']

'uct.ac.za'
```

The Regex Version

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('@([^ ]*)',lin)
print y['uct.ac.za']
```

```
'@([^ ]*)'
```

Look through the string until you find an at-sign

The Regex Version

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('@([^ ]*)',lin)
print y['uct.ac.za']
```

```
'@([^ ]*)'
```

Match non-blank character

Match many of them

The Regex Version

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('@([^ ]*)',lin)
print y['uct.ac.za']
```

$$\underbrace{\texttt{'@([\^{}]*)'}}_{\text{Extract the non-blank characters}}$$

Even Cooler Regex Version

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('^From .*@([^ ]*)',lin)
print y['uct.ac.za']
```

```
'^From .*@([^ ]*)'
```

Starting at the beginning of the line, look for the string 'From '

Even Cooler Regex Version

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('^From .*@([^ ]*)',lin)
print y['uct.ac.za']
```

```
'^From .*@([^ ]*)'
```

Skip a bunch of characters, looking for an at-sign

Even Cooler Regex Version

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('^From .*@([^ ]*)',lin)
print y['uct.ac.za']
```

$$\overset{\displaystyle \texttt{'\^{}From .*@}\,\texttt{([\^{}]*)'}}{}$$

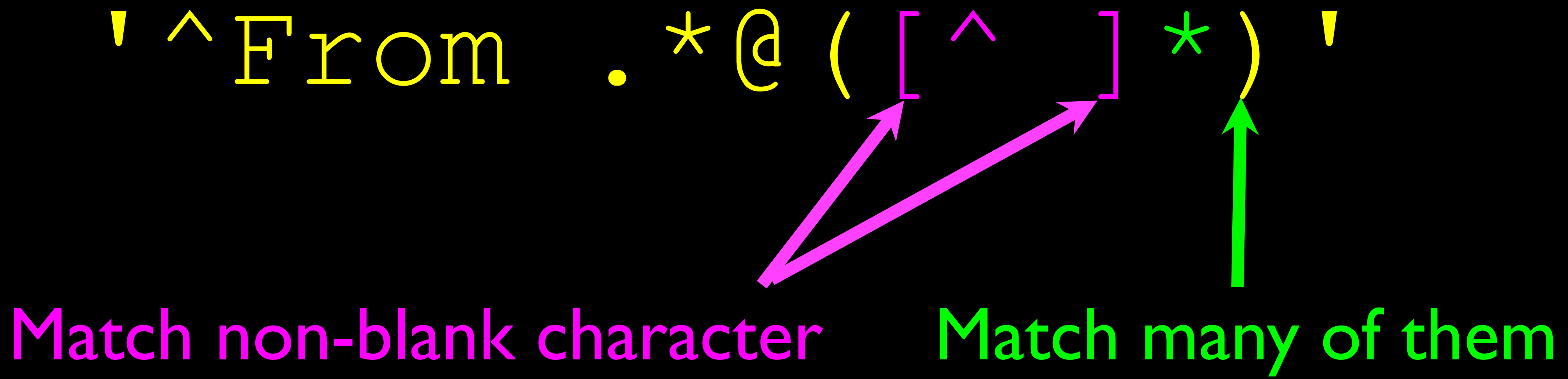
```
'^From .*@([^ ]*)'
```

$\uparrow$ Start 'extracting'

Even Cooler Regex Version

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('^From .*@([^ ]*)',lin)
print y['uct.ac.za']
```



```
'^From .*@([^ ]*)'
```

Even Cooler Regex Version

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('^From .*@([^ ]*)',lin)
print y['uct.ac.za']
```

```
'^From .*@([^ ]*)'
```

Stop 'extracting'

Spam Confidence

```
import re
hand = open('mbox-short.txt')
numlist = list()
for line in hand:
    line = line.rstrip()
    stuff = re.findall('^X-DSPAM-Confidence: ([0-9.]+)', line)
    if len(stuff) != 1 :  continue
    num = float(stuff[0])
    numlist.append(num)
print 'Maximum:', max(numlist)
```

```
python ds.py
Maximum: 0.9907
```

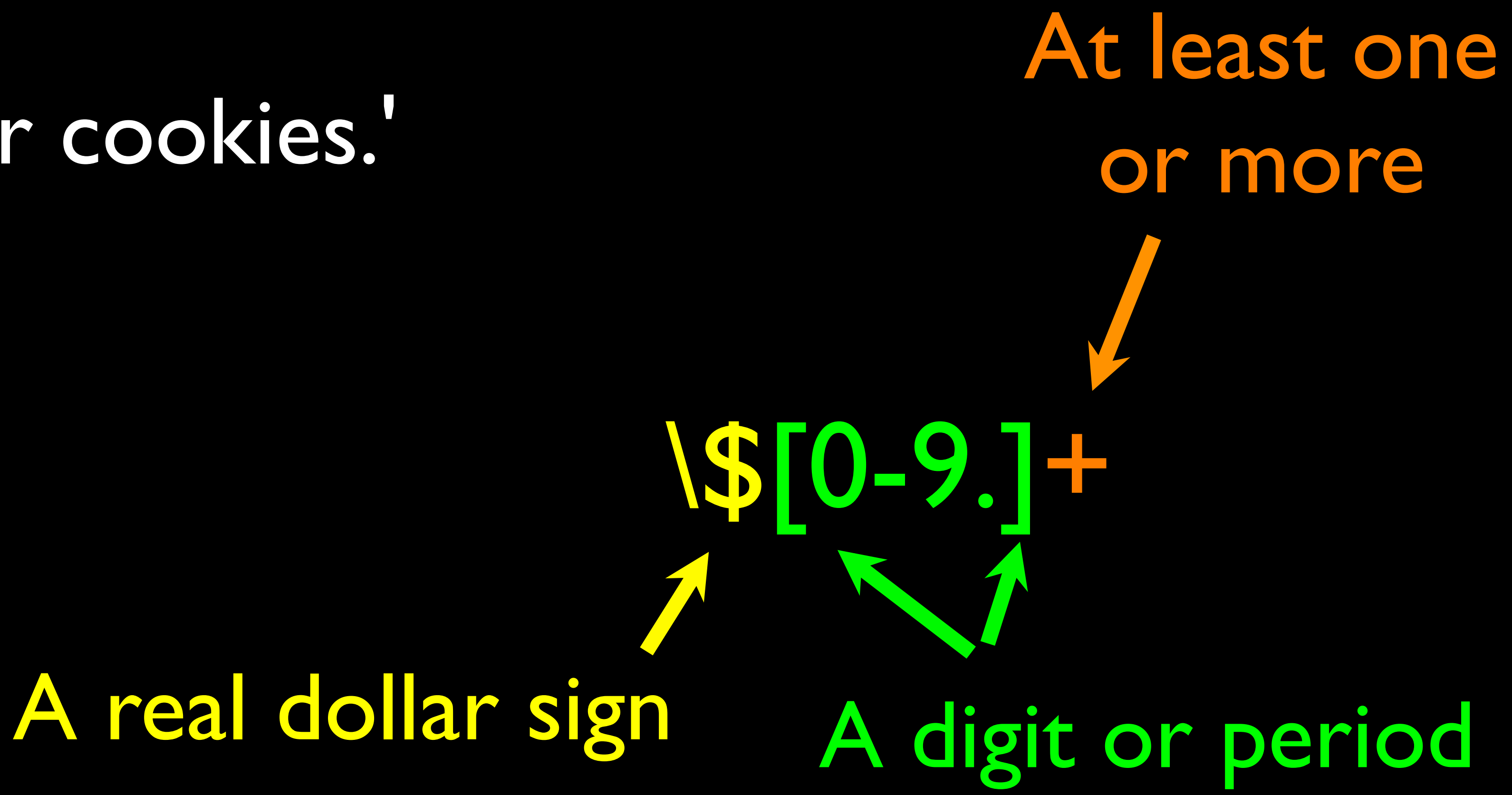
Regular Expression Quick Guide

`^`	Matches the beginning of a line
`$`	Matches the end of the line
`.`	Matches any character
`\s`	Matches whitespace
`\S`	Matches any non-whitespace character
`*`	Repeats a character zero or more times
`*?`	Repeats a character zero or more times (non-greedy)
`+`	Repeats a chracter one or more times
`+?`	Repeats a character one or more times (non-greedy)
`[aeiou]`	Matches a single character in the listed set
`[^XYZ]`	Matches a single character not in the listed set
`[a-z0-9]`	The set of characters can include a range
`(`	Indicates where string extraction is to start
`)`	Indicates where string extraction is to end

Escape Character

- If you want a special regular expression character to just behave normally (most of the time) you prefix it with '\'

```
>>> import re
>>> x = 'We just received $10.00 for cookies.'
>>> y = re.findall('\$[0-9.]+',x)
>>> print y
['$10.00']
```



Summary

- Regular expressions are a cryptic but powerful language for matching strings and extracting elements from those strings
- Regular expressions have special characters that indicate intent