

# Regular Expressions

## Chapter 11

Python for Everybody  
[www.py4e.com](http://www.py4e.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](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](http://en.wikipedia.org/wiki/Regular_expression)

Regular expression - Wikipedia, the free encyclopedia

W http://en.wikipedia.org/wiki/Regular\_expression Reader Google

More than 100 matches regular Done

Log in / create account

WIKIPEDIA The Free Encyclopedia

Main page Contents Featured content Current events Random article Donate to Wikipedia

Interaction Help About Wikipedia Community portal Recent changes Contact Wikipedia

Article Discussion Read Edit View history Search

## 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!

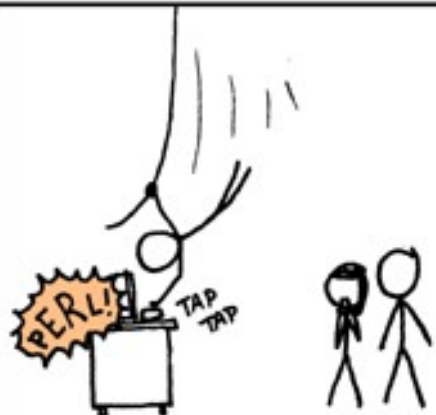
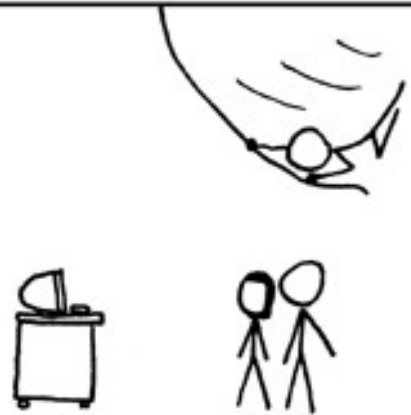


IT'S HOPELESS!

EVERYBODY STAND BACK.



I KNOW REGULAR EXPRESSIONS.



<http://xkcd.com/208/>

# Regular Expression Quick Guide

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

Match the start of the  
line

Many  
times

^ X . \* :

Match any character

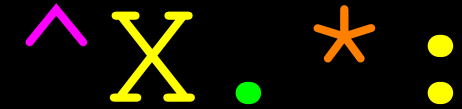
# Fine-Tuning Your Match

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-: Very short
```

Match the start  
of the line

Many  
times

  
^X.\*:

Match any character

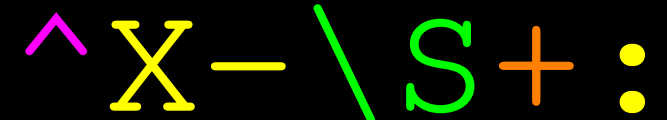
# Fine-Tuning Your Match

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-: Very Short
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

- `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 substrings that match the regular expression

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

Why not 'From:' ?

First character in the match is an F

Last character in the match is a :

One or more characters



# 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 greedy

`^F . + ? :`

First character in the match is an F

Last character in the match is a :

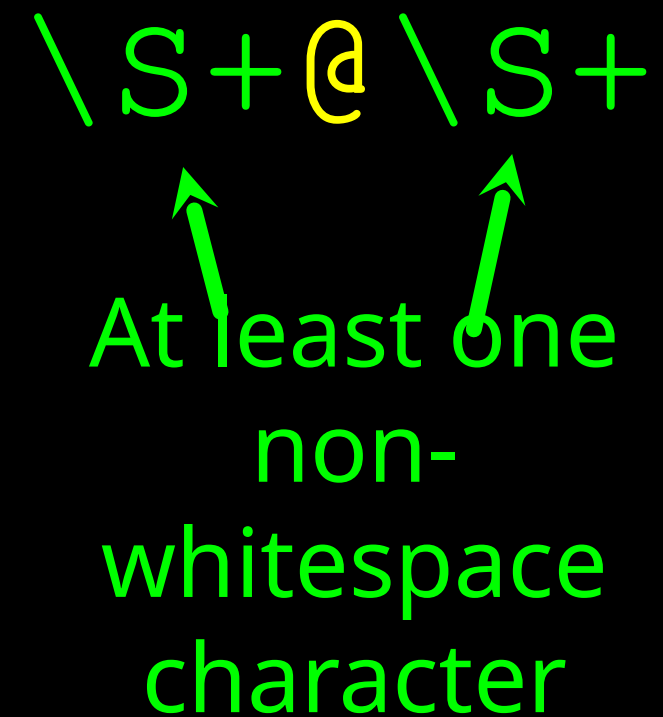
# Fine-Tuning String Extraction

You can refine the match for `re.findall()` and separately determine which portion of the match is to be extracted by using parentheses

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



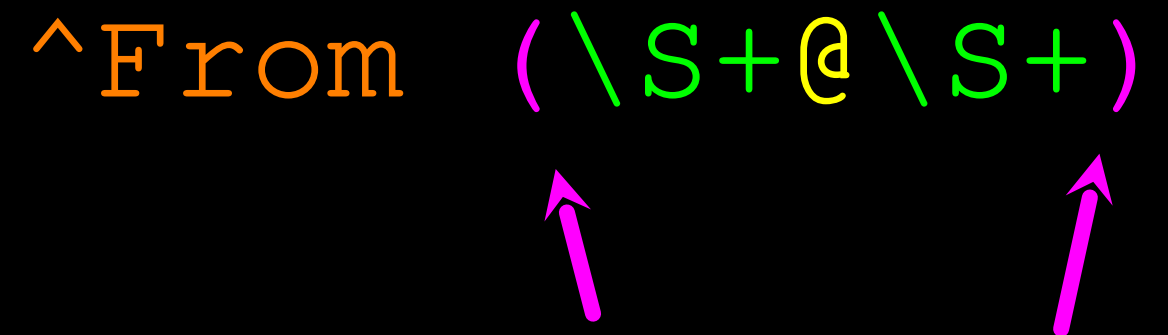
# Fine-Tuning String Extraction

**Parentheses** 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']
```

^From (\S+@\S+)



# String Parsing Examples...

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

21 ↓ 31 ↓

```
>>> 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 Pattern

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'
```

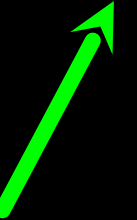
# The Regex Version

```
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

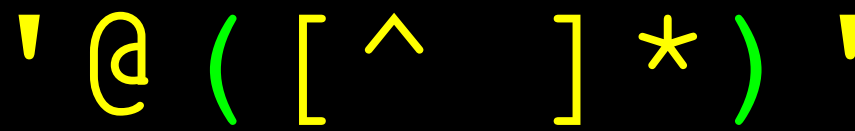
# The Regex Version

```
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']
```

'@([ ^ ]\*)'



Extract the non-blank characters

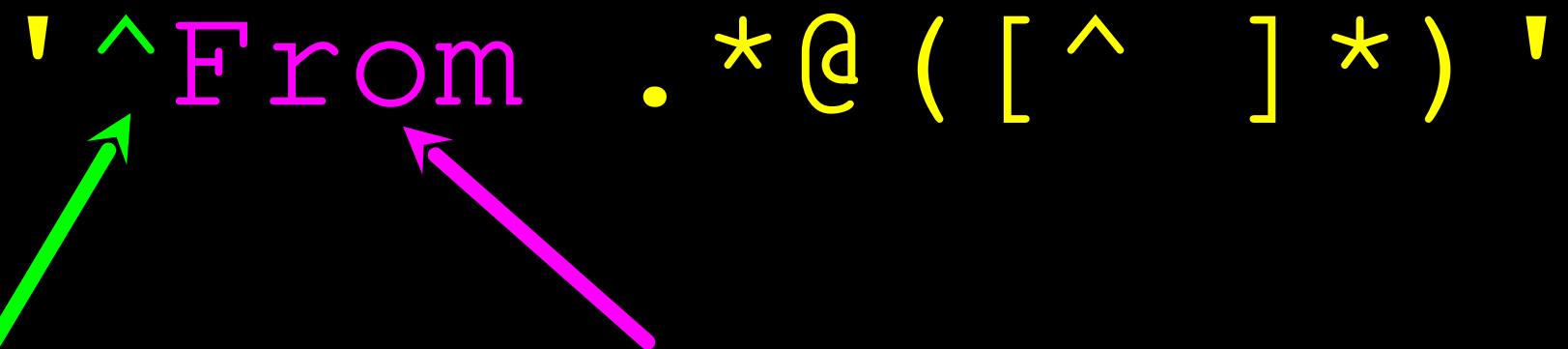
# Even Cooler Regexp Version

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

A diagram illustrating the regex pattern '^From .\*@([ ^ ]\*)'. A green arrow points from the '^' character to the text 'Starting at the beginning of the line, look for the string 'From''. A purple arrow points from the 'From' text to the 'F' character in the pattern.

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']
```

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



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 .\*@([ ^ ]+)'

Match non-blank  
character

Match many of  
them

# 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**

X-DSPAM-Confidence: 0.8475

# 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']
```

At least one  
or more



\\$ [0-9.] +

A real dollar sign

A digit or period

# 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



# Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance ( [www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information and [open.umich.edu](http://open.umich.edu) and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translations here