



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



Understanding Regular Expressions

Very powerful and quite cryptic

Fun once you understand them

Regular expressions are a language unto themselves

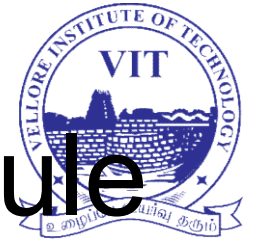
A language of “marker characters” - programming with characters

It is kind of an “old school” language - compact



Regular Expression Quick Guide

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

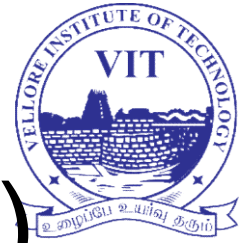


The Regular Expression Module

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()` 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



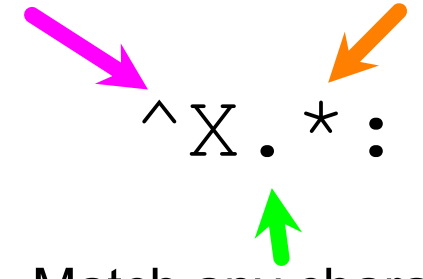
Depending on how “clean” your data is and the purpose of your application, you may want to narrow your match down a bit

```
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

Match any character

A diagram illustrating the components of the regular expression ^X.*:. A pink arrow points from the text 'Match the start of the line' to the '^' character. An orange arrow points from the text 'Many times' to the '*' character. A green arrow points from the text 'Match any character' to the '.' character. The characters are arranged as '^ X . * :' with spaces between them.

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

$\wedge X - \backslash S + :$

Match any non-whitespace character



Matching and Extracting Data

`re.search()` returns a True/False depending on whether the string matches the regular expression

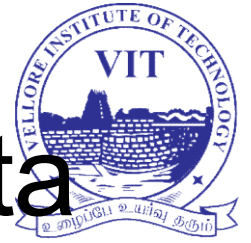
If we actually want the matching strings to be extracted, we use `re.findall()`

`[0-9]+`



One or more digits

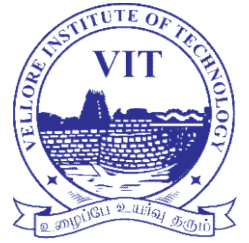
```
>>> import re
>>> x = 'My 2 favorite numbers are 19 and 42'
>>> y = re.findall('[0-9]+',x)
>>> print(y)
['2', '19', '42']
```



Matching and Extracting Data

When we use `re.findall()`, it returns a list of zero or more sub-strings 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:' ?

One or more
characters

^ F . + :

First character in
the match is an F

Last character in the
match is a :

A diagram illustrating the greedy matching of the regular expression pattern `^F.+:` against the string `'From: Using the : character'`. The pattern is shown as `^ F . + :`. A green arrow points from the text 'First character in the match is an F' to the `F` in the pattern. A yellow arrow points from the text 'Last character in the match is a :' to the `:` in the pattern. An orange arrow points from the text 'One or more characters' to the `+` in the pattern.



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:']
```

[^]F . + ? :

One or more
characters but
not greedy

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

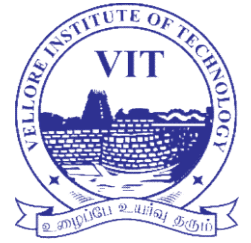
```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

```
>>> 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...





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**