

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)
[]
```

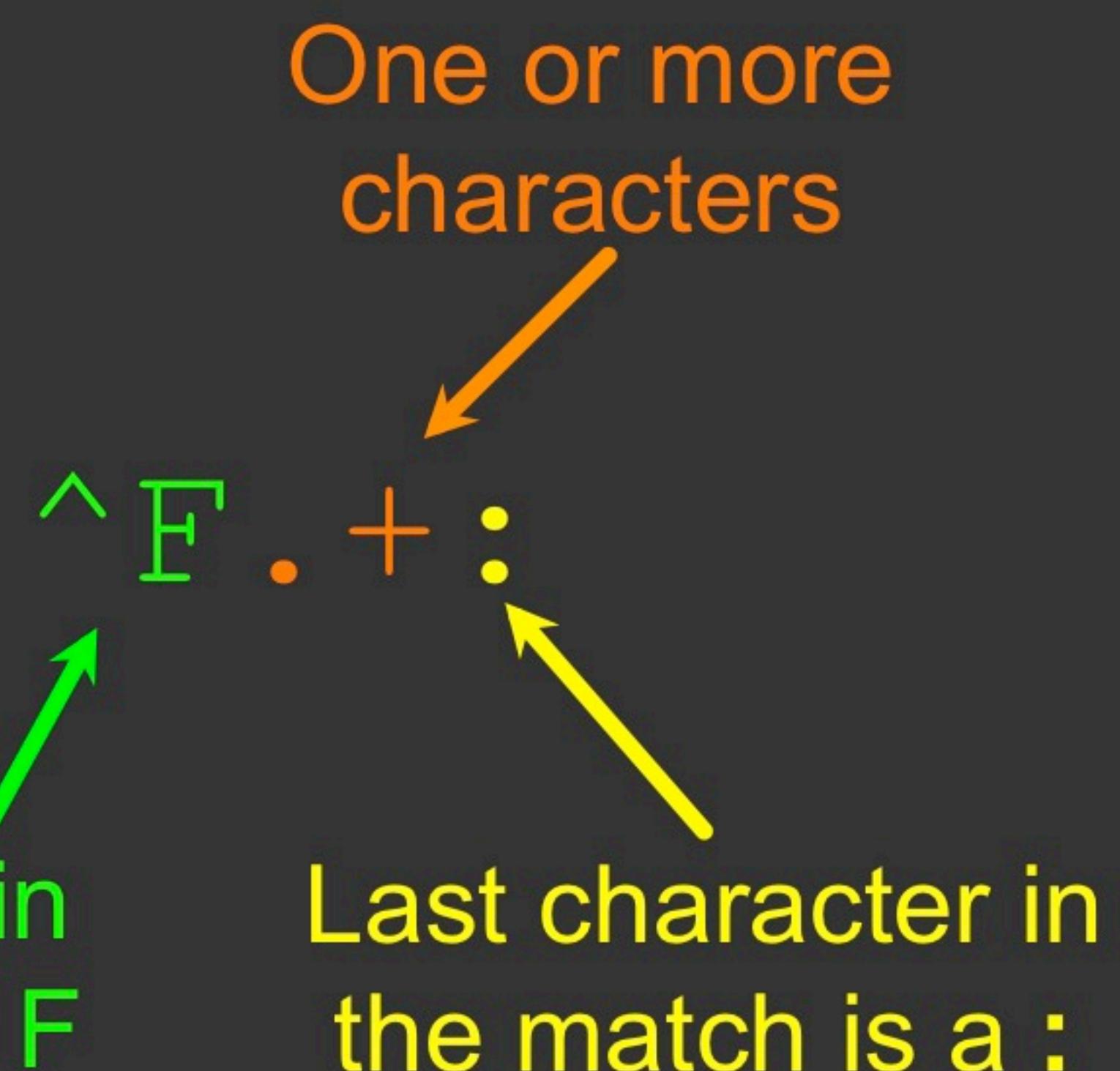
Warning: Greedy Matching

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



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:']
```

First character in
the match is an F

One or more
characters but
not greedy

^F . + ?:

↑
↑
Last character in
the match is a :



Advanced RegEx...

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-
white space
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+)



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  
>>> spos = data.find(' ',atpos)  
>>> print(spos)  
31  
>>> host = data[atpos+1 : spos]  
>>> 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('@([^\s]*)',lin)
print(y)
```

['uct.ac.za']

'@([^\s]*)'



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('@([^\s]*)',lin)
print(y)
```

['uct.ac.za']

'@([^\s]*)'

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('@([^\s]*)',lin)
print(y)
```

['uct.ac.za']

'@([^\s]*)'

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 .*@[^\s]*',lin)
print(y)
```

```
['uct.ac.za']
```

```
'^From .*@[^\s]*'
```

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 .*@[^\s]*',lin)
print(y)
```

```
['uct.ac.za']
```

```
'^From .*@[^\s]*'
```

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 .*@[^\s]*',lin)
print(y)
```

['uct.ac.za']

'^From .*@[^\s]*'



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 .*@[^\s]*',lin)
print(y)
```

['uct.ac.za']

'^From .*@[^\s]*'

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 .*@[^\n ]+',lin)
print(y)
```

```
['uct.ac.za']
```

```
'^From .*@[^\n ]+'
```



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))
```

X-DSPAM-Confidence: 0.8475

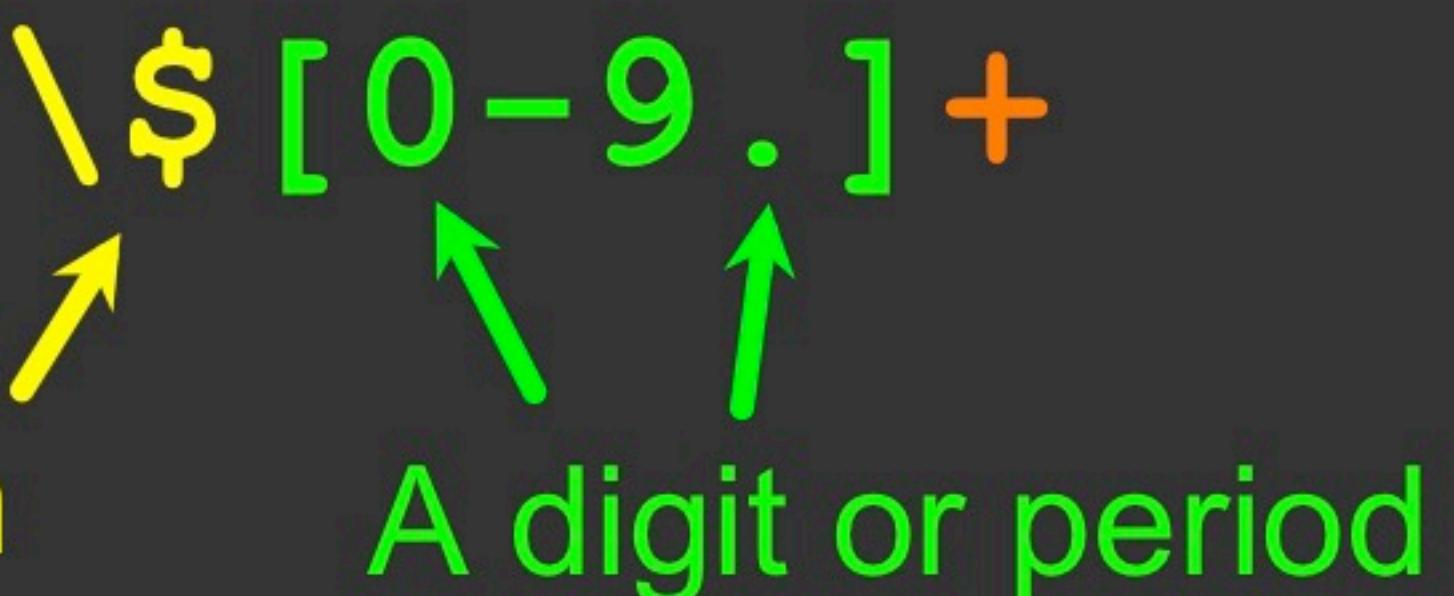
python ds.py
Maximum: 0.9907

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']
```

A real dollar sign

At least one
or more

\\$ [0-9.] +
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) of the University of Michigan School of Information and 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

