LECTURE 8
REGULAR EXPRESIONS

Fundamentals of Programming - COMP1005

Semester 2, 2019

Department of Computing

Curtin University

Copyright Warning

REGULAR EXPRESSIONS

Fundamentals of Programming

Lecture 7

The story so far…

- We've worked out how to read and write files
- We've read csv files with text and numbers
- So far they've been nicely formatted
  …. well-behaved
- What if the formatting isn't consistent?
- How do we clean up files without having to manually fix everything?
- **Regular expressions**

Regular Expressions

- We've been able to do string matching to check for values
- And string matching to pick delimiters (',')
- There are times we want more flexibility than an exact string
  - e.g. phone numbers – have various formats, need to extract the numbers

```
+61 8 9266 1000
(08) 9266 1000
(08) 92661000
08 9266 1000
08 92661000
92661000
```


https://docs.python.org/3/howto/regex.html

Wildcards

- When we search our directories, we can use wildcards to match multiple files:
  - `grep matplotlib *.py`
  - *.py matches to heatsource.py, growth.py etc.
- * matches to zero or more characters
  - `heat*.py` matches to heat.py, heatsource.py
- Regular expressions work in a similar way

Metacharacters

- Most letters and characters match to themselves
  - "test" will match to "test"
- Metacharacters have special meaning:
  - . ^ $ * + ? { } [ ] \ | ( )
- [ ] is a set of characters to match (class)
  - [cbm]at will match to cat and bat and mat
- ^ gives the opposite (complement)
  - [^5] will match anything but 5

Metacharacters

- \ gives special sequences, or overrides a \
  - \[is just a [character
- Special sequences
  - \d – matches to any decimal digit == [0-9]
  - \D – matches any non-decimal digit [^0-9]
  - \s – matches any whitespace character [\t\n\r\f\v]
  - \S – matches any non-whitespace character
  - \w – matches any alphanumeric char [a-zA-Z0-9]
  - \W – matches any non-alphanumeric character

Metacharacters

- . matches anything other than newline

- Examples
  - .at – matches anything with a character followed by "at"
  - [0-9][a-z] – matches any digit followed by a lower case character
  - [0-9]\s[a-z] – matches any digit followed by a whitespace charcater, followed by a lower case character

Metacharacters - repetition

- * matches to zero or more repeats of the previous colour or class
  - ca*t matches to ct, cat, caaat, caaaaaat etc
- + matches to one or more repeats
  - ca+t matches cat, caaaaat but not ct
- {m,n} matches at least m repeats and at most n repeats
  - a/{1,3}b matches to a/b, a//b, a///b but not ab or a////b
  - missing m or n defaults to 0 or infinity respectively

Using regular expressions

- `import re`
- `p = re.compile(r'ab*')`
  - r indicates a raw string – doesn't interpret \n etc
- Methods
  - match() – match to beginning of string
  - search() – match to anywhere in the string
  - findall() – returns a list of matches
  - finditer() – returns an iterator of matches

Regular expressions

- The result of matching is a match object
- We can get different parts of the object:
  - group() – the string that was matched
  - start() – starting position of the match
  - end() – ending position of the match
  - span() – a tuple containing start and end

- We'll work through all of this in the pracs…

Regular expressions – phone numbers

```
import re

phoneRegex = re.compile(r'''(
    (\d{2}|\(\d{2}\))
    (\s|\.)?
    (\d{4}\s\d{4})
    )''', re.VERBOSE)

fileobj = open('phone.txt')
data = fileobj.readlines()

for number in data:
    number = number.strip()
    mo = phoneRegex.search(number)
    if mo:
        print(number + ' found ' + mo.group())
```

Summary

- Revisited formatted text files (e.g. csv) to store and load data
- Learnt about using regular expressions to check and clean data
- Will explore and analyse real-world datasets (in pracs)