

# Regular expressions

Week 1, day 3-4

# The Project

- Suggestions next week, but own ideas preferred. Topics chosen & approved by end of 3rd week.
- A “substantial piece of software”.
- Assessed by a 10-15 minutes presentation - motivation, demonstration, design - and the source code.
- Can work alone or in pairs.

# The trouble with strings

- Powerful methods for searching and replacing
- But all require an inflexible search string
  - `string.find (substr)` returns the index at which `substr` is found.
  - `string.count (substr)` returns the number of occurrences of `substr`.
  - `string.replace (substr1, substr2)` returns a copy of with all instances of `substr1` have been replaced with `substr2`.

# The trouble with strings 2

- Arguments are strict and literal, no allowance for even minor variation
  - To find `TATA` or `TATTA` or `TAATA` within a string requires three separate calls to `string.find()`
  - To count the number of times the letter `C` occurs within a string, regardless of upper or lower case, requires two separate calls to `string.count()`.
  - To replace all consecutive stretches of whitespace in a string with a single space (e.g. 3 spaces and a tab in a row become one space), there is no function that can reliably handle all cases.

# Regular expressions

- “Regexes”
- A syntax (language) for programming machines that recognise text
- Simple patterns (plain alphanumerics) match only themselves

```
\bw[a-z]*
```

```
search_str = """A  
friend is someone who  
will help you move.A  
real friend is someone  
who will help you move  
a body."""
```

```
# finds 'who' and 'will'
```

# Repetition / count operators

- `?` : 0 or 1 time
- `*` : 0 or more times
- `+` : 1 or more times
- `{x,y}` : from `x` to `y` times
  - `{x,}` : `x` or more times
  - `{:y}` : `y` or less times
  - `{z}` : `z` times exactly

```
bre?  
bre*  
bre+  
bre{2,3}
```

```
br  
bre  
bree  
breeee  
breeeee
```

```
# GREEDY!
```

# Position meta- characters

- Match not characters but positions and places
- `^` : beginning of string
- `$` : end of string
- `\b` : word boundary

```
bre^  
^bre  
bre$  
bre*\b  
  
br  
bre  
bree  
breee  
breee foo
```

# Regexes in Python (finally)

```
>>> import re
>>> myRegex = re.compile (r'\d+')
>>> myRegex.findall ('12 drummers')
['12']

# 3 search functions of the form:
# x = string.method (str [, start] [, end])
```

# Regex in Python (details)

- Many functions but 4 main ones:
  - 3 search functions:
    - `string.findall (str) => [matching substrings]`
    - `string.search (str) => Match for first match`
    - `string.finditer (str) => iterator over Matches for all matches`
  - 1 substitution function:
    - `string.sub(search_str, replace_str) => new string`

# The Match object

- Stores where and what was matched in `group()`, `start()`, `end()` methods
- Also stores results of subpatterns in `group(n)`, `start(n)`, `end(n)`
- Can get all matches with `groups()`

# Subpatterns

```
>>> pat = re.compile( r'name:\s+(\w+)\s+number:(\d+)' )
>>> m = pat.match('xyzy name: rod number:123') # match at start
>>> print m
None
>>> m = pat.search('xyzy name: rod number:123') # search in string
>>> m.groups()
('rod', '123')
```

# Verbose regexes

```
pattern = r"""
^                # beginning of string
M{0,4}          # thousands - 0 to 4 M's
(CM|CD|D?C{0,3}) # hundreds - 900 (CM), 400 (CD), 0-300 (0 to 3 C's),
#                # or 500-800 (D, followed by 0 to 3 C's)
(XC|XL|L?X{0,3}) # tens - 90 (XC), 40 (XL), 0-30 (0 to 3 X's),
#                # or 50-80 (L, followed by 0 to 3 X's)
(IX|IV|V?I{0,3}) # ones - 9 (IX), 4 (IV), 0-3 (0 to 3 I's),
#                # or 5-8 (V, followed by 0 to 3 I's)
$
"""
```

# A handy test function

```
def testRegex (re_str, search_str):  
    """  
    Test regex by running on search string & returning  
    results.  
  
    Just a time saver.  
    """  
    import re  
    theRe = re.compile (re_str)  
    return theRe.findall (search_str)
```

# Examples I

- Match a positive integer
- Match a birthday, in the form `YYYY-MM-DD` (assume it's after 1900).
- Find two given words (e.g. `foo` & `bar`) separated by 1 or 2 other words.

# Solutions I

```
# positive integer  
[1-9]\d*
```

```
# birthdate after 1900, YYYY-MM-DD  
(19|20)\d\d\-(0[1-9]|1[012])\-(0[1-9]|12)[0-9]|3[01])
```

```
# 'foo' & 'bar' separated by 1 or 2 words  
foo\s+(\w+\s+){1,2}bar
```

# Examples 2

- Match a valid email address
- Matching flanking whitespace
- Extract the home phone from strings  
work: 123-5678 HOME: 987 1331  
Can be all upper- or all lowercase.

# Solutions 2

```
# email address  
[1-9]\d*
```

```
# match flanking whitespace  
^[ \s]+|[ \s]+$
```

```
# get home phone number  
(home|HOME):\s*(\d[ \s\d]*\d+)
```

# Examples 3

# what do these mean?

```
[^\w\s]|[0-9]
```

```
^(([A-Z][0-9]{5})|([A-Z]{2}[0-9]{6}))$
```

```
^accession[^\d-9 \.:\; \?]*[A-Z]{1,2}  
[0-9]{3,7}[ \.:\;?]+.*
```

# Further study

- The Python library reference on regexes: <http://python.org/doc/current/lib/module-re.html>
- How to do Regexes in Python: <http://www.amk.ca/python/howto/regex/>
- A tutorial, examples and reference sheet at <http://www.regular-expressions.info>