```python
# Search for lines that contain 'From'
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('From:', line):
        print(line)
```

```python
# Search for lines that start with 'From'
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:', line):
        print(line)
```

```python
# Search for lines that start with 'F', followed by
# 2 characters, followed by 'm:'
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^F..m:', line):
        print(line)
```

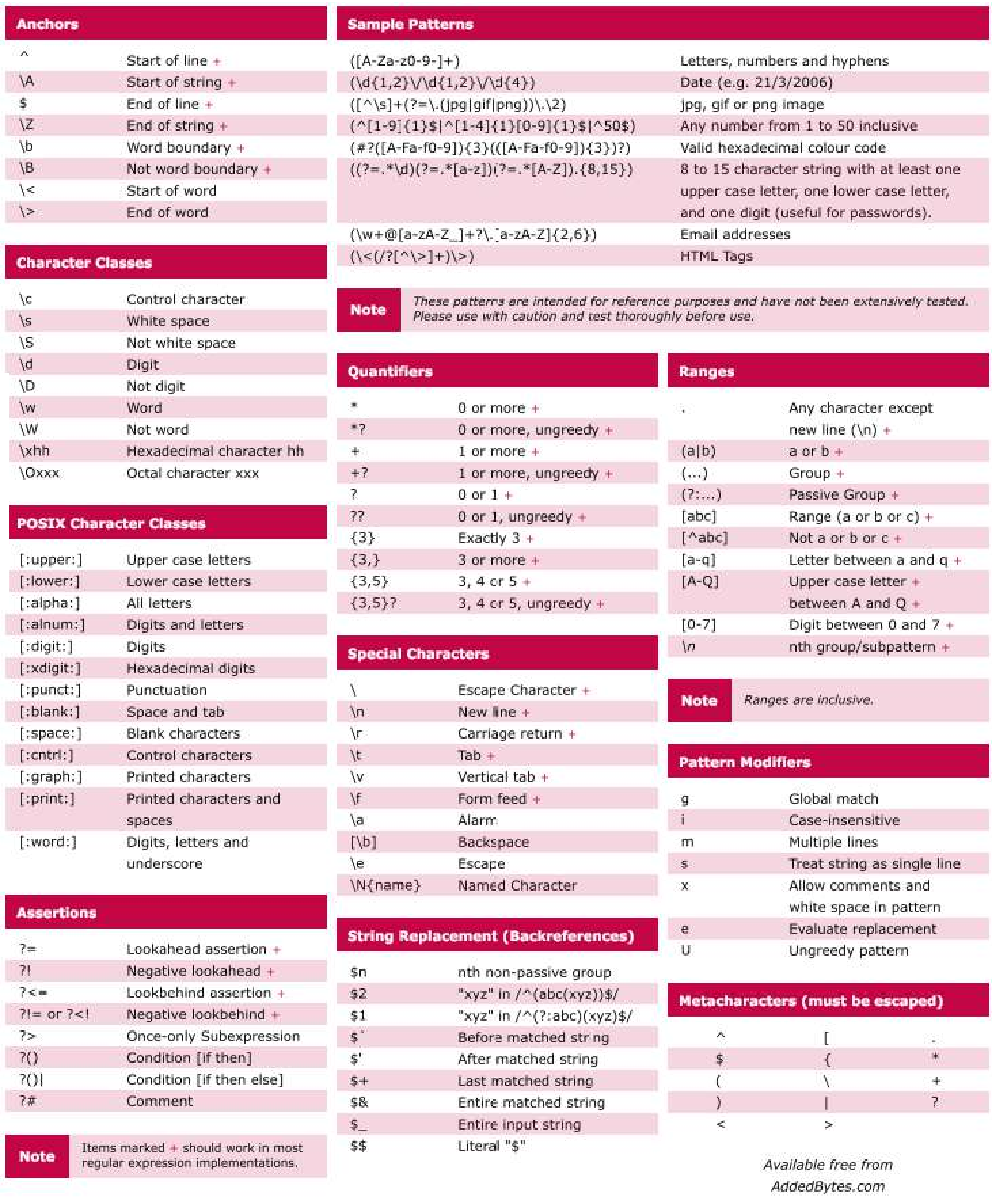
```python
# Search for lines that start with From and have an at sign
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:.+@', line):
        print(line)
```

```
['wagnermr@iupui.edu']
['cwen@iupui.edu']
['<postmaster@collab.sakaiproject.org>']
['<200801032122.m03LMFo4005148@nakamura.uits.iupui.edu>']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['apache@localhost)']
['source@collab.sakaiproject.org;']
```

Some of our email addresses have incorrect characters like "<" or ";" at the beginning or end. Let's declare that we are only interested in the portion of the string that starts and ends with a letter or a number.

To do this, we use another feature of regular expressions. Square brackets are used to indicate a set of multiple acceptable characters we are willing to consider matching. In a sense, the \S is asking to match the set of "non-whitespace characters". Now we will be a little more explicit in terms of the characters we will match.

Here is our new regular expression: `[a-zA-Z0-9]\S*@\S*[a-zA-Z]`



Anchors	
^	Start of line +
\A	Start of string +
$	End of line +
\Z	End of string +
\b	Word boundary +
\B	Not word boundary +
\<	Start of word
\>	End of word

Character Classes	
\c	Control character
\s	White space
\S	Not white space
\d	Digit
\D	Not digit
\w	Word
\W	Not word
\xhh	Hexadecimal character hh
\Oxxx	Octal character xxx

POSIX Character Classes	
[:upper:]	Upper case letters
[:lower:]	Lower case letters
[:alpha:]	All letters
[:alnum:]	Digits and letters
[:digit:]	Digits
[:xdigit:]	Hexadecimal digits
[:punct:]	Punctuation
[:blank:]	Space and tab
[:space:]	Blank characters
[:cntrl:]	Control characters
[:graph:]	Printed characters
[:print:]	Printed characters and spaces
[:word:]	Digits, letters and underscore

Assertions	
?=	Lookahead assertion +
?!	Negative lookahead +
?<=	Lookbehind assertion +
?!= or ?<!	Negative lookbehind +
?>	Once-only Subexpression
?()	Condition [if then]
?()|	Condition [if then else]
?#	Comment

Note *Items marked + should work in most regular expression implementations.*

Sample Patterns	
([A-Za-z0-9-]+)	Letters, numbers and hyphens
(\d{1,2}\/\d{1,2}\/\d{4})	Date (e.g. 21/3/2006)
([^\s]+(?=\.(jpg|gif|png))\.\2)	jpg, gif or png image
(^[1-9]{1}$\|^[1-4]{1}[0-9]{1}$|^50$)	Any number from 1 to 50 inclusive
(#?([A-Fa-f0-9]){3}(([A-Fa-f0-9]){3})?)	Valid hexadecimal colour code
((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,15})	8 to 15 character string with at least one upper case letter, one lower case letter, and one digit (useful for passwords).
(\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,6})	Email addresses
(\<(/?[^\>]+)\>)	HTML Tags

Note *These patterns are intended for reference purposes and have not been extensively tested. Please use with caution and test thoroughly before use.*

Quantifiers	
*	0 or more +
*?	0 or more, ungreedy +
+	1 or more +
+?	1 or more, ungreedy +
?	0 or 1 +
??	0 or 1, ungreedy +
{3}	Exactly 3 +
{3,}	3 or more +
{3,5}	3, 4 or 5 +
{3,5}?	3, 4 or 5, ungreedy +

Special Characters	
\	Escape Character +
\n	New line +
\r	Carriage return +
\t	Tab +
\v	Vertical tab +
\f	Form feed +
\a	Alarm
[\b]	Backspace
\e	Escape
\N{name}	Named Character

String Replacement (Backreferences)	
$n	nth non-passive group
$2	"xyz" in /^(abc(xyz))$/
$1	"xyz" in /^(?:abc)(xyz)$/
$`	Before matched string
$'	After matched string
$+	Last matched string
$&	Entire matched string
$_	Entire input string
$$	Literal "$"

Ranges	
.	Any character except new line (\n) +
(a|b)	a or b +
(...)	Group +
(?:...)	Passive Group +
[abc]	Range (a or b or c) +
[^abc]	Not a or b or c +
[a-q]	Letter between a and q +
[A-Q]	Upper case letter + between A and Q +
[0-7]	Digit between 0 and 7 +
\n	nth group/subpattern +

Note *Ranges are inclusive.*

Pattern Modifiers	
g	Global match
i	Case-insensitive
m	Multiple lines
s	Treat string as single line
x	Allow comments and white space in pattern
e	Evaluate replacement
U	Ungreedy pattern

Metacharacters (must be escaped)		
^	[	.
$	{	*
(	\	+
)	|	?
<	>	

Available free from AddedBytes.com

```
import re
s = 'A message from csev@umich.edu to cwen@iupui.edu'
lst = re.findall('\S+@\S+', s)
print(lst)
                    ['csev@umich.edu', 'cwen@iupui.edu']
```

```
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
# Search for lines that start with 'Details: rev='
# followed by numbers and '.'
# Then print the number if it is greater than zero
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^Details:.*rev=([0-9.]+)', line)
    if len(x) > 0:
        print(x)
```

```
# Search for lines that have an at sign between characters
# The characters must be a letter or number
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('[a-zA-Z0-9]\S+@\S+[a-zA-Z]', line)
    if len(x) > 0:
        print(x)
```

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
# Search for lines that start with From and a character
# followed by a two digit number between 00 and 99 followed by ':'
# Then print the number if it is greater than zero
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^From .* ([0-9][0-9]):', line)
    if len(x) > 0: print(x)
```

```
# Search for lines that start with 'X' followed
# by any non whitespace characters and ':'
# followed by a space and any number.
# The number can include a decimal.
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^X\S*: [0-9.]+', line):
        print(line)
```

```
# Search for lines that start with 'X' followed by any
# non whitespace characters and ':' followed by a space
# and any number. The number can include a decimal.
# Then print the number if it is greater than zero.
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^X\S*: ([0-9.]+)', line)
    if len(x) > 0:
        print(x)
```

```
import re
x = 'We just received $10.00 for cookies.'
y = re.findall('\$[0-9.]+',x)
```

Since we prefix the dollar sign with a backslash, it actually matches the dollar

We will use `urllib` to read the page and then use `BeautifulSoup` to extract the `href` attributes from the anchor (`a`) tags.

```
import urllib.request, urllib.parse, urllib.error
from bs4 import BeautifulSoup
import ssl

# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

url = input('Enter - ')
html = urllib.request.urlopen(url, context=ctx).read()
soup = BeautifulSoup(html, 'html.parser')

# Retrieve all of the anchor tags
tags = soup('a')
for tag in tags:
    print(tag.get('href', None))
```

The program prompts for a web address, then opens the web page, reads the data and passes the data to the BeautifulSoup parser, and then retrieves all of the anchor tags and prints out the `href` attribute for each tag.

```
Enter - https://docs.python.org
genindex.html
py-modindex.html
https://www.python.org/
#
whatsnew/3.6.html
whatsnew/index.html
tutorial/index.html
```

```
# Retrieve all of the anchor tags
tags = soup('a')
for tag in tags:
    # Look at the parts of a tag
    print('TAG:', tag)
    print('URL:', tag.get('href', None))
    print('Contents:', tag.contents[0])
    print('Attrs:', tag.attrs)
```

```
Enter - http://www.dr-chuck.com/page1.htm
TAG: <a href="http://www.dr-chuck.com/page2.htm">
Second Page</a>
URL: http://www.dr-chuck.com/page2.htm
Content: ['\nSecond Page']
Attrs: [('href', 'http://www.dr-chuck.com/page2.htm')]
```

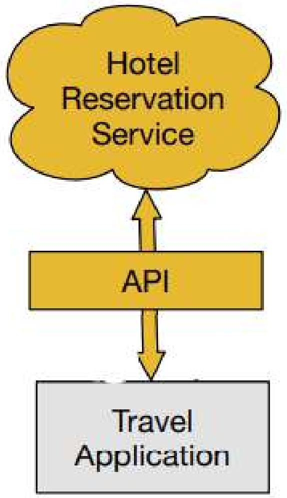
Parsing XML

```
import xml.etree.ElementTree as ET

data = '''
<person>
  <name>Chuck</name>
  <phone type="intl">
    +1 734 303 4456
  </phone>
  <email hide="yes" />
</person>'''

tree = ET.fromstring(data)
print('Name:', tree.find('name').text)
print('Attr:', tree.find('email').get('hide'))
```

```
Name: Chuck
Attr: yes
```



```
import xml.etree.ElementTree as ET

input = '''
<stuff>
  <users>
    <user x="2">
      <id>001</id>
      <name>Chuck</name>
    </user>
    <user x="7">
      <id>009</id>
      <name>Brent</name>
    </user>
  </users>
</stuff>'''

stuff = ET.fromstring(input)
lst = stuff.findall('users/user')
print('User count:', len(lst))

for item in lst:
    print('Name', item.find('name').text)
    print('Id', item.find('id').text)
    print('Attribute', item.get('x'))
```

```
User count: 2
Name Chuck
Id 001
Attribute 2
Name Brent
Id 009
Attribute 7
```

JSON

In JSON, we simply have key-value pairs. Also the XML "person" tag is gone, replaced by a set of outer curly braces.

```
{
  "name" : "Chuck",
  "phone" : {
    "type" : "intl",
    "number" : "+1 734 303 4456"
   },
   "email" : {
     "hide" : "yes"
   }
}
```

```
import json
data = '''
[
  { "id" : "001",
    "x" : "2",
    "name" : "Chuck"
  } ,
  { "id" : "009",
    "x" : "7",
    "name" : "Brent"
  }
]'''
info = json.loads(data)
print('User count:', len(info))
for item in info:
    print('Name', item['name'])
    print('Id', item['id'])
    print('Attribute', item['x'])
```

```
User count: 2
Name Chuck
Id 001
Attribute 2
Name Brent
Id 009
Attribute 7
```

```
class PartyAnimal:
   x = 0

   def __init__(self):
     print('I am constructed')

   def party(self) :
     self.x = self.x + 1
     print('So far',self.x)

   def __del__(self):
     print('I am destructed', self.x)
```

```
an = PartyAnimal()              I am constructed
an.party()                      So far 1
an.party()                      So far 2
an = 42                         I am destructed 2
print('an contains',an)         an contains 42

an = PartyAnimal()              I am constructed
                                I am destructed 0
```

```
class PartyAnimal:
   x = 0
   name = ''
   def party(self) :
     self.x = self.x + 1
     print(self.name,'party count',self.x)

s = PartyAnimal('Sally')
j = PartyAnimal('Jim')
```

```
TypeError: PartyAnimal() takes no arguments
```

```
class PartyAnimal:
   x = 0
   name = ''
   def __init__(self, nam):
     self.name = nam
     print(self.name,'constructed')

   def party(self) :
     self.x = self.x + 1
     print(self.name,'party count',self.x)

s = PartyAnimal('Sally')
j = PartyAnimal('Jim')

s.party()
j.party()
s.party()
```

```
Sally constructed
Jim constructed
Sally party count 1
Jim party count 1
Sally party count 2
```

For this example, we move our `PartyAnimal` class into its own file. Then. we can 'import' the `PartyAnimal` class in a new file and extend it

```
from party import PartyAnimal

class CricketFan(PartyAnimal):
   points = 0
   def six(self):
      self.points = self.points + 6
      self.party()
      print(self.name,"points",self.points)

s = PartyAnimal("Sally")
s.party()
j = CricketFan("Jim")
j.party()
j.six()
print(dir(j))
```

```
Sally constructed
Sally party count 1
Jim constructed
Jim party count 1
Jim party count 2
Jim points 6
['__class__', '__delattr__', ... '__weakref__',
'name', 'party', 'points', 'six', 'x']
```

Just as the `def` keyword does not cause function code to be executed, the `class` keyword does not create an object. Instead, the `class` keyword defines a template indicating what data and code will be contained in each object of type `PartyAnimal`.