More Regular Expressions

Nov 5 2015

Class Today

- Continue with regular expressions
- Match iterators
- More special characters
- Working with match *groups*

- Let's talk about the project first

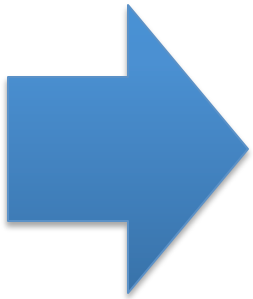
Calendar

Sun	Mon	Tues	Wed	Thurs	Fri	Sat
11/1	11/2	11/3	11/4	11/5	11/6	11/7
				HW 3-1 out		
		Work on your projects!				
11/8	11/9	11/10	11/11	11/12	11/13	11/14
		HW 3-1 due				
		Work on your projects!				
11/15 **Project due**						

Loading dictionary words into Python

dictionary.txt

```
Aam\n
Aaronic\n
Aaronical\n
Ab\n
Abaca\n
Abacinate\n
```



dict

Word	Freq.
Aam	1
Aaronic	1
Aaronical	1
Ab	1
Abaca	1
Abacinate	1

Loading dictionary words into Python

```
# read contents of dictionary
file = open('dictionary.txt', 'r')
contents = file.read()
file.close()

# get list of words
words = contents.split('\n')

# add all words into dict
# note that dictionaries have unique keys (not duplicated)
dict = {}
for word in words[:10]:
    dict[word] = 1
```

To search in the dictionary:

```
key in dict    (evaluates to True or False)
```

Class Today

- Continue with Regular Expressions
- Match iterators
- More special characters
- Working with match *groups*

Good News!

Regular Expressions

`The qu1ck brown fox jumped over the lazy d0g.`

Special Syntax	Meaning	R.E Example	Result
[ ]	Match anything between brackets	`[qjd]u`	`qu ju`

Regular Expressions

`The qu1ck brown fox jumped over the lazy d0g.`

Special Syntax	Meaning	R.E Example	Result
[ ]	Match anything between brackets	`[qjd]u`	`qu ju`
\w	Match any alphanumeric	`o\w`	`ow ox ov`

Regular Expressions

The qu1ck brown fox jumped over the lazy d0g.

Special Syntax	Meaning	R.E Example	Result
[ ]	Match anything between brackets	`[qjd]u`	`qu ju`
\w	Match any alphanumeric	`o\w`	`ow ox ov`
\s	Match any whitespace	`e\s\w`	`'e q' 'e l'`

Regular Expressions

`The qu1ck brown fox jumped over the lazy d0g.`

Special Syntax	Meaning	R.E Example	Result
[ ]	Match anything between brackets	`[qjd]u`	`qu ju`
\w	Match any alphanumeric	`o\w`	`ow ox ov`
\s	Match any whitespace	`e\s\w`	`'e q' 'e l'`
\d	Match any digit	`\d\w`	`1c 0g`

Regular Expressions

`The qu1ck brown fox jumped over the lazy d0g.`

Special Syntax	Meaning	R.E Example	Result
[ ]	Match anything between brackets	`[qjd]u`	`qu ju`
\w	Match any alphanumeric	`o\w`	`ow ox ov`
\s	Match any whitespace	`e\s\w`	`'e q' 'e l'`
\d	Match any digit	`\d\w`	`1c 0g`
+	Match **one or more** of the previous things.	`\s\w+\s`	`' qu1ck ' ' fox ' ' over ' ' lazy '`

Regular Expressions

`The qu1ck brown fox jumped over the lazy d0g.`

Special Syntax	Meaning	R.E Example	Result
[ ]	Match anything between brackets	`[qjd]u`	`qu ju`
\w	Match any alphanumeric	`o\w`	`ow ox ov`
\s	Match any whitespace	`e\s\w`	`'e q' 'e l'`
\d	Match any digit	`\d\w`	`1c 0g`
+	Match **one or more** of the previous things.	`\s\w+\s`	`qu1ck fox over lazy`
*	Match **zero or more** of the previous things.	`\w\d*\w`	`Th qu 1c br ow fo ju mp ed ov er th la zy d0g`

Regular Expressions

The qu1ck brown fox jumped over the lazy d0g.

Special Syntax	Meaning	R.E Example	Result
[ ]	Match anything between brackets	`[qjd]u`	`qu ju`
\w	Match any alphanumeric	`o\w`	`ow ox ov`
\s	Match any whitespace	`e\s\w`	`'e q' 'e l'`
\d	Match any digit	`\d\w`	`1c 0g`
+	Match **one or more** of the previous things.	`\s\w+\s`	`qu1ck fox over lazy`
*	Match **zero or more** of the previous things.	`\w\d*\w`	`Th qu 1c br ow fo ju mp ed ov er th la zy d0g`
.	Match any character		

Regular Expressions

The qu1ck brown fox jumped over the lazy d0g.

Special Syntax	Meaning	R.E Example	Result
[ ]	Match anything between brackets	`[qjd]u`	`qu ju`
\w	Match any alphanumeric	`o\w`	`ow ox ov`
\s	Match any whitespace	`e\s\w`	`'e q' 'e l'`
\d	Match any digit	`\d\w`	`1c 0g`
+	Match **one or more** of the previous things.	`\s\w+\s`	`qu1ck fox over lazy`
*	Match **zero or more** of the previous things.	`\w\d*\w`	`Th qu 1c br ow fo ju mp ed ov er th la zy d0g`
.	Match any character	`.`	

Regular Expressions

The qu1ck brown fox jumped over the lazy d0g.

Special Syntax	Meaning	R.E Example	Result
[ ]	Match anything between brackets	`[qjd]u`	`qu ju`
\w	Match any alphanumeric	`o\w`	`ow ox ov`
\s	Match any whitespace	`e\s\w`	`'e q' 'e l'`
\d	Match any digit	`\d\w`	`1c 0g`
+	Match **one or more** of the previous things.	`\s\w+\s`	` qu1ck fox over lazy`
*	Match **zero or more** of the previous things.	`\w\d*\w`	`Th qu 1c br ow fo ju mp ed ov er th la zy d0g`
.	Match any character	`.`	`T h e ' ' q u ...`

Regular Expressions

The qu1ck brown fox jumped over the lazy d0g.

Syntax	Meaning	R.E Example	Result
[ ]	Match anything between brackets	`[qjd]u`	`qu ju`
\w	Match any alphanumeric	`o\w`	`ow ox ov`
\s	Match any whitespace	`e\s\w`	`'e q' 'e l'`
\d	Match any digit	`\d\w`	`1c 0g`
+	Match **one or more** of the previous things.	`\s\w+\s`	` qu1ck fox over lazy`
*	Match **zero or more** of the previous things.	`\w\d*\w`	`Th qu 1c br ow fo ju mp ed ov er th la zy d0g`
.	Match any character	`\s.+\s`	`' qu1ck brown fox jumped over the lazy '`

Regular Expressions

The qu1ck brown fox jumped over th lazy d0g.

Do Task 2

Special Syntax	Meaning	R.E	
[ ]	Match anything between brackets	`[qjd]u`	qu ju
\w	Match any alphanumeric	`o\w`	ow ox ov
\s	Match any whitespace	`e\s\w`	'e q' 'e l'
\d	Match any digit	`\d\w`	1c 0g
+	Match **one or more** of the previous things.	`\s\w+\s`	qu1ck fox over lazy
*	Match **zero or more** of the previous things.	`\w\d*\w`	Th qu 1c br ow fo ju mp ed ov er th la zy d0g
.	Match any character	`\s.+\s`	qu1ck brown fox

Regular Expressions

Just the beginning… see the 'PythonRE' link for lots more:

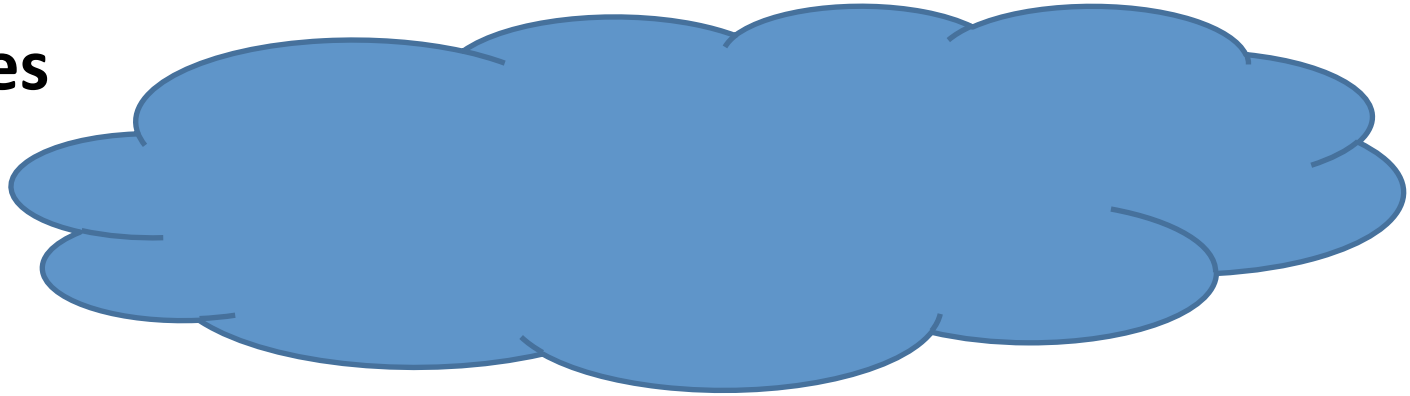
Pattern	Description
^	Matches beginning of line.
$	Matches end of line.
.	Matches any single character except newline.
[...]	Matches any single character in brackets.
[^...]	Matches any single character not in brackets
re*	Matches 0 or more occurrences of preceding expression.
re+	Matches 1 or more occurrence of preceding expression.
re?	Matches 0 or 1 occurrence of preceding expression.
re{ n}	Matches exactly n number of occurrences of preceding expression.
re{ n,}	Matches n or more occurrences of preceding expression.
re{ n, m}	Matches at least n and at most m occurrences of preceding expression.
a| b	Matches either a or b.
(re)	Groups regular expressions and remembers matched text.
…	…

Data Structures

Lists

content	‘a’	‘b’	‘c’	‘d’	‘e’	‘f’	‘g’	‘h’	‘i’	‘j’
indices	0	1	2	3	4	5	6	7	8	9

Dictionaries



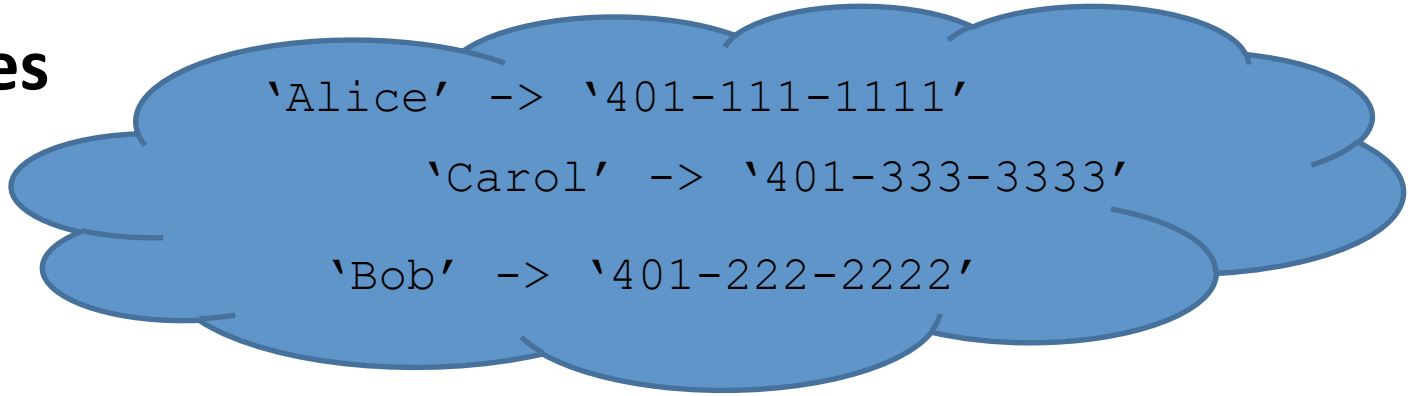
Data Structures

Lists

content	‘a’	‘b’	‘c’	‘d’	‘e’	‘f’	‘g’	‘h’	‘i’	‘j’
indices	0	1	2	3	4	5	6	7	8	9

Dictionaries

keys & values



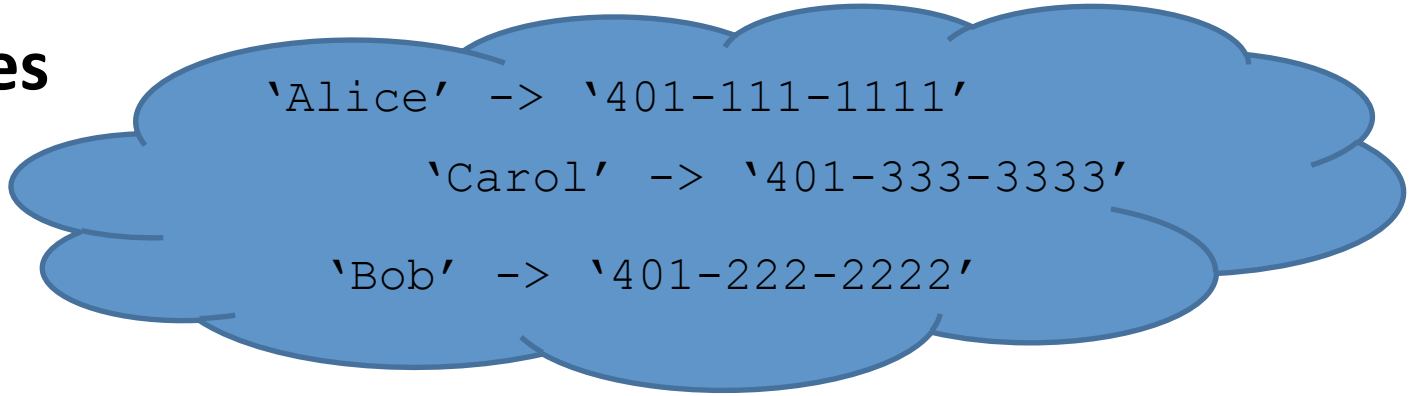
Data Structures

Lists

content	‘a’	‘b’	‘c’	‘d’	‘e’	‘f’	‘g’	‘h’	‘i’	‘j’
indices	0	1	2	3	4	5	6	7	8	9

Dictionaries

keys & values



Iterators

Match Objects

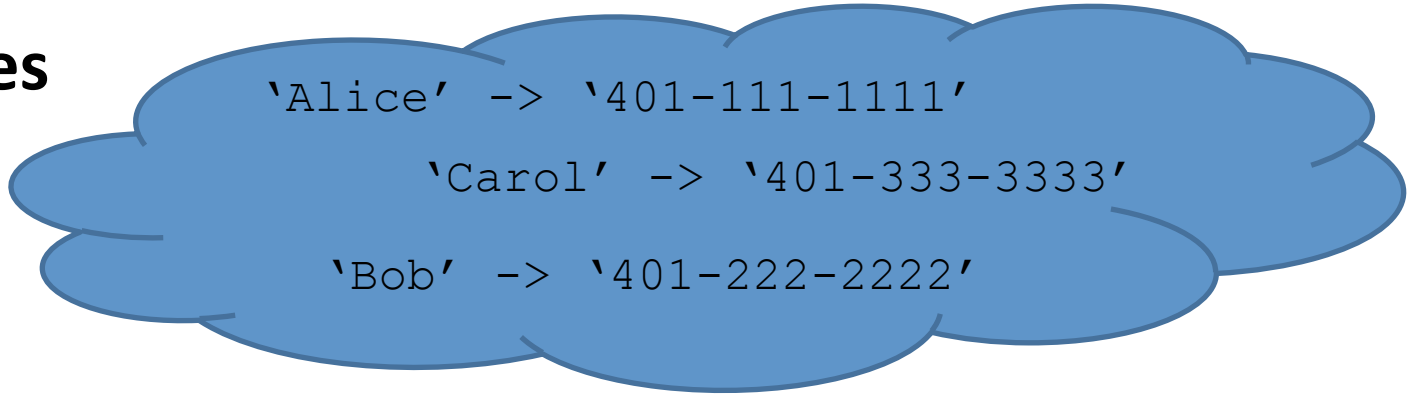
Data Structures

Lists

content	‘a’	‘b’	‘c’	‘d’	‘e’	‘f’	‘g’	‘h’	‘i’	‘j’
indices	0	1	2	3	4	5	6	7	8	9

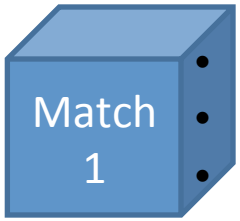
Dictionaries

keys & values



Iterators

Match Objects



- Matched String
- Matched String Start
- Matched String End

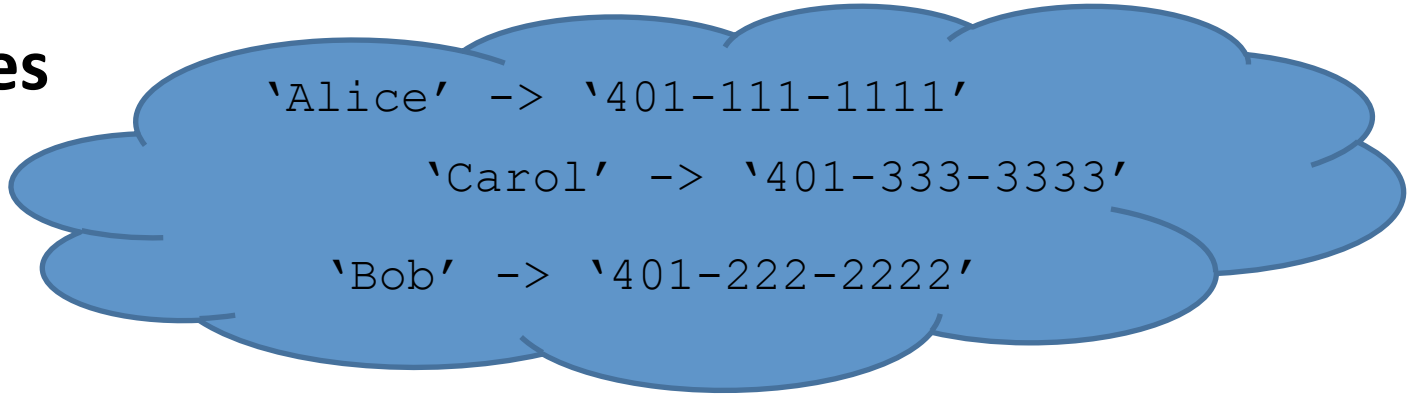
Data Structures

Lists

content	‘a’	‘b’	‘c’	‘d’	‘e’	‘f’	‘g’	‘h’	‘i’	‘j’
indices	0	1	2	3	4	5	6	7	8	9

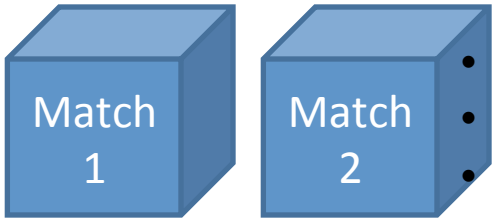
Dictionaries

keys & values



Iterators

Match Objects



- Matched String
- Matched String Start
- Matched String End

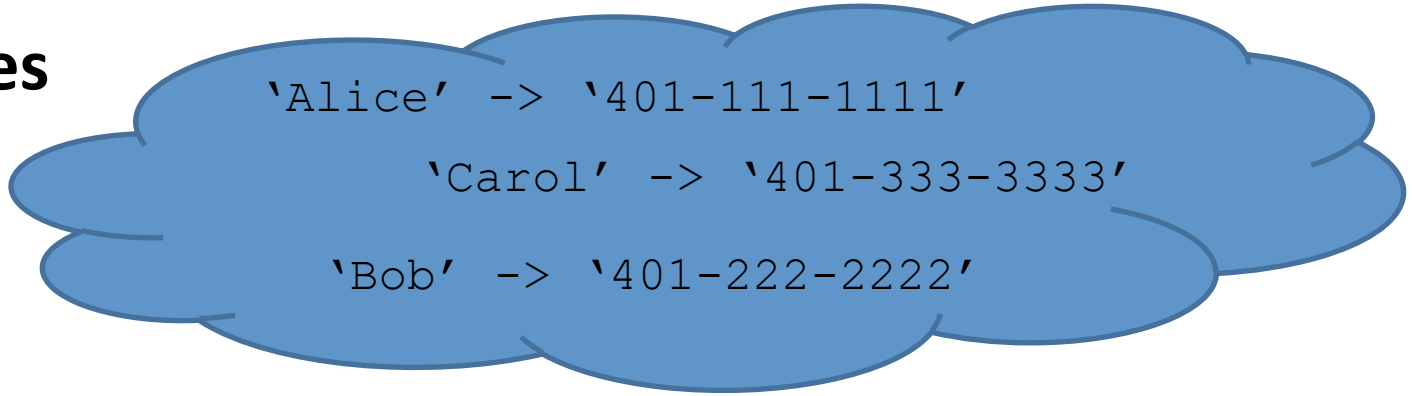
Data Structures

Lists

content	‘a’	‘b’	‘c’	‘d’	‘e’	‘f’	‘g’	‘h’	‘i’	‘j’
indices	0	1	2	3	4	5	6	7	8	9

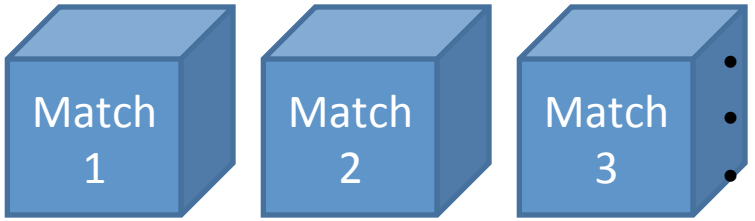
Dictionaries

keys & values



Iterators

Match Objects



- Matched String
- Matched String Start
- Matched String End

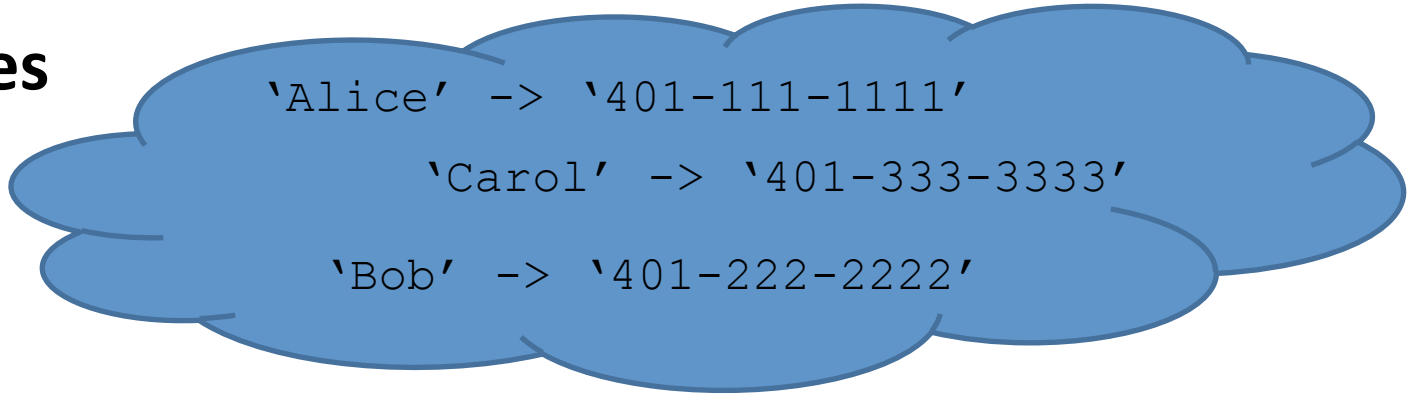
Data Structures

Lists

content	'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'	'i'	'j'
indices	0	1	2	3	4	5	6	7	8	9

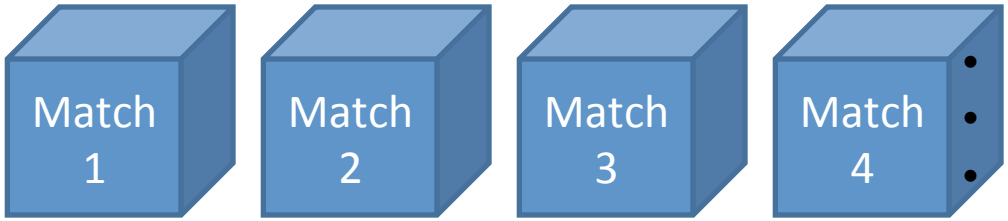
Dictionaries

keys & values



Iterators

Match Objects



- Matched String
- Matched String Start
- Matched String End

Iterators

The cat in the hat sat on a mat

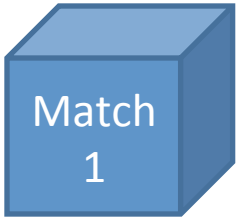
Regular Expression:‘\wat’

Iterators

The cat in the hat sat on a mat

Regular Expression:'\wat'

Iterator

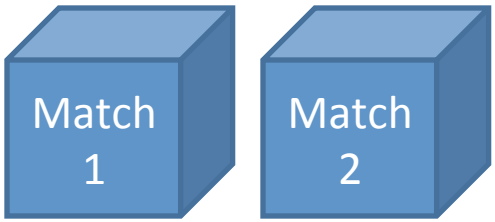


Iterators

The cat in the hat sat on a mat

Regular Expression:'\wat'

Iterator

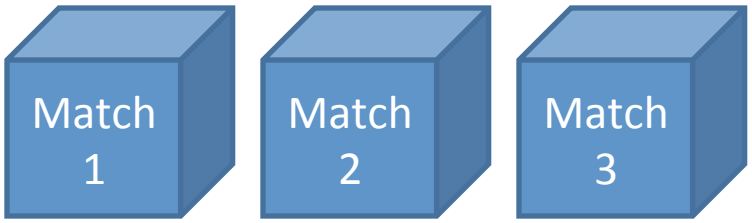


Iterators

The cat in the hat sat on a mat

Regular Expression:'\wat'

Iterator

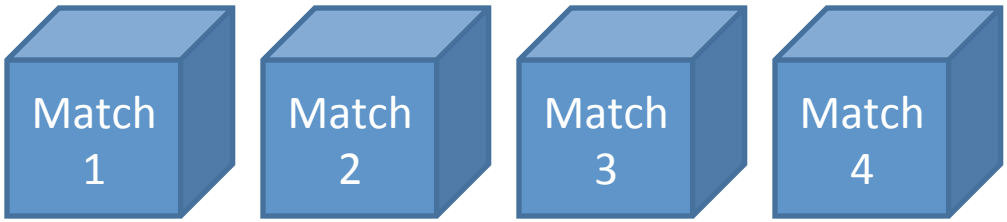


Iterators

The cat in the hat sat on a mat

Regular Expression:'\wat'

Iterator



Using Regular Expressions in Python

```
def printRegEx(regex,myStr):
    '''Prints all occurrences of the regular expression.'''

    myIter = re.finditer(regex,myStr) # Iterator!

    # This iterator contains MATCHES of the regex.
    # The following functions work on regex matches:
    #  group(0): returns the string that matches the regex
    #  start(0): the starting position of the string in myStr
    #  end(0): the ending position of the string in myStr

    # For loops work on iterators in addition to lists
    # Each match of the regex in myStr is stored in
    # a variable called 'occ'
    for occ in myIter:
        print('matches',occ.group(0), \
            'at positions',occ.start(0),'-',occ.end(0))
    return
```

Search using RegEx in P

- To use, type `import re` Python code, then functio
  - `re.search('a\w+a',`
    - Evaluates to `True`
      - You can find 'anica' in the string
  - `re.search('x\w+i', 'mechanical')`
    - Evaluates to `False`
      - You cannot find any substring starting with 'x' and finishing with 'i' in the string

In groups, do Task 1

Match using RegEx in P

- To use, type `import re` Python code, then functio
  - `re.match('a\w+a', 'l`
    - Evaluates to `False`
      - The word is not in the specified form
  - `re.match('a\w+a', 'abracadabra')`
    - Evaluates to `True`
      - The word is in the specified form

In groups, do Task 2

Task 2

```
for word in myList:
    # print word only if it rhymes with ‘ping pong’
```

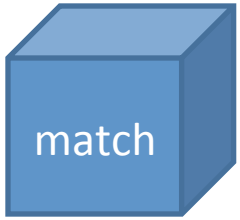
Task 2

```
for word in myList:
    match = re.match('\w*ing\s*\w*ong', word)
    if match:
        print word
```

Search and Match return MatchObjects

```
match = re.search('\s\w+\s', 'the cat jumped')
```

```
if match :
    print match.group(0)
```



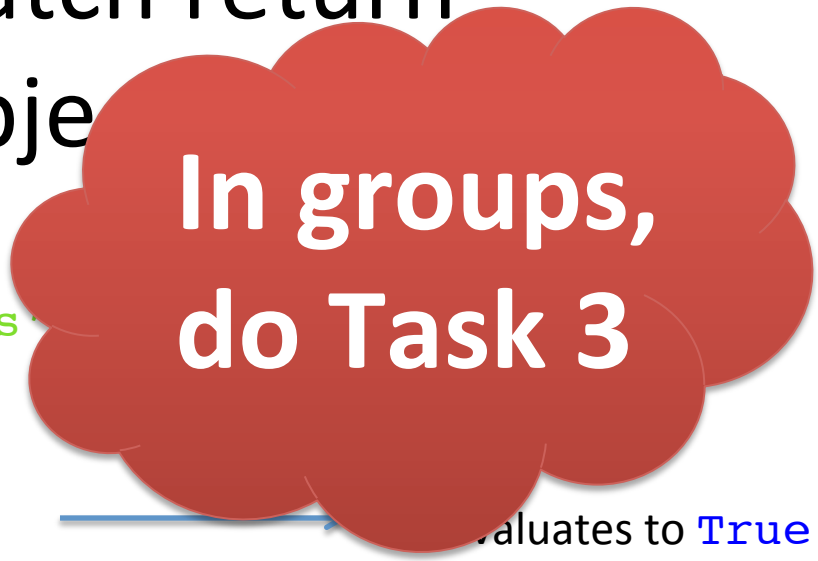
Evaluates to `True`

Gives you back what matched

`' cat '`

Not really, you don't want the spaces

Search and Match return MatchObje




```
match = re.search('\s(\w+)\s
```

valuates to `True`

```
if match :
```

```
print match.group(0)
print match.group(1)
```

Gives you back

0: the whole match

1…n: the corresp. parenthesis match

```
' cat '
'cat'
```

Task 3

```
for word in myList:
    match = re.match('(\w*ing)\s*(\w*ong)', word)
    if match:
        print 'Whole match: ', match.group(0)
        print 'First sub-match: ', match.group(1)
        print 'Second sub-match: ', match.group(2)
```

Match Iterators

- Before, we just checked to see whether a pattern existed in the text...
- Or if the text started with a pattern

- Let's now do some computing with those parts of the text that match

Match Iterators

- Before, we just checked to see whether a pattern existed in the text...
- Or if the text started with a pattern

- Let's now do some computing with those parts of the text that match

Data Structures

Lists

content	'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'	'i'	'j'
indices	0	1	2	3	4	5	6	7	8	9

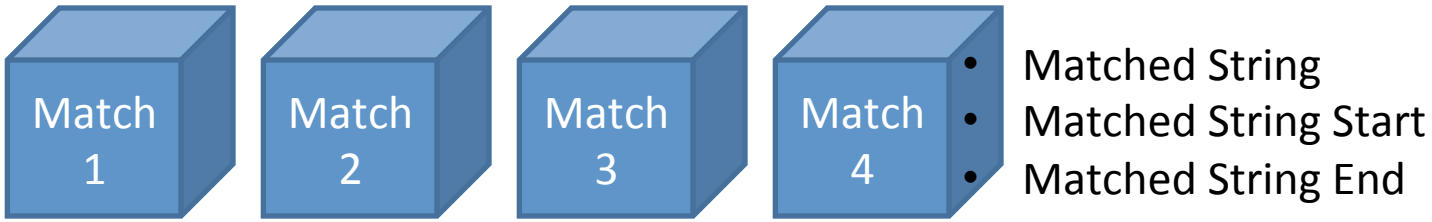
Dictionaries

keys & values

Key	Value
`'alice'`	`'401-111-1111'`
`'carol'`	`'401-222-2222'`
`'bob'`	`'401-333-3333'`

Iterators

Match Objects



Matching iterative


```
def printRegEx(regex,myStr):
    '''Prints all occurrences of the regular expres     '''

    match_iterator = re.finditer(regex,myStr) # Iterator!

    # This iterator contains MATCHES of the regex.
    # The following functions work for each match:
    #  group(0): returns the string that matches the regex
    #  start(0): the starting position of the string in myStr
    #  end(0): the ending position of the string in myStr

    # For loops work on iterators in addition to lists
    # Each match of the regex in is stored
    # in the variable 'match', in sequence
    for match in match_iterator:
        print('matches',match.group(0), \
            'at positions',match.start(0), '-', match.end(0))
    return
```

Pattern	Description
\s	Match a whitespace character (except newline, if you use the "re.MULTILINE" option).
\w	Match a letter, number, or underscore character.
\d	Match a digit.
^	Matches beginning of string (or beginning of line with the "re.MULTILINE" option).
$	Matches end of string (or end of line with the "re.MULTILINE" option).
.	Matches any single character except newline (also including newline with the "re.MULTILINE" option).
[aeiou]	Matches any single character in brackets.
[A-Z]	Matches any single character in the range indicated in the brackets.
[^aeiou]	Matches any single character not in brackets.
[aeiou]*	Matches 0 or more occurrences of preceding expression.
[aeiou]+	Matches 1 or more occurrence of preceding expression.
[aeiou]?	Matches 0 or 1 occurrence of preceding expression.
[aeiou]{n}	Matches exactly n number of occurrences of preceding expression.
[aeiou]{n,}	Matches n or more occurrences of preceding expression.
[aeiou]{n, m}	Matches at least n and at most m occurrences of preceding expression.
a|b	Matches either a or b.
([aeiou]+)	Groups regular expressions and remembers matched text.
…	…

Check how we got the words from the *Online Plain Text English Dictionary*

```
# opens output file for writing
outputFile = open(outputFilename, 'w')

# for each letter
for letter in 'abcdefghijklmnopqrstuvwxyz':
  # download dictionary contents for the letter
  # urlLetter is the URL 'constructed" based on it
  remoteFile = urllib.request.urlopen(urlLetter)
  contents = remoteFile.read().decode('utf-8')
  remoteFile.close()

  # collect all the words (made only by characters)
  # that are inside '<B> </B>' and write to the output file
  for match in re.finditer('<[Bb]>(\w+)</[Bb]>', contents):
    outputFile.write(match.group(1))
    outputFile.write('\n')

outputFile.close()
```

group(1)…group(n) also work for each match in the iterator

re_tester.py

```
def showMatches(pattern, text):
    print('Text: ' + text)
    print('Patt: ' + pattern)
    match_iterator = re.finditer(pattern, text)
    for match in match_iterator:
        print('------- New Match -------')
        print('  match.start() = ' + str(match.start()))
        print('  match.end()   = ' + str(match.end()))
        print('  Whole match: ' + match.group(0))
        if match.lastindex is not None:
            print('  Match groups:')
            for i in range(1, match.lastindex+1):
                print('    match.group(' + str(i) + '): '
                      + match.group(i))
```

Match Iterators

- Compare the output of these expressions:
  - `showMatches('a\w+b.z', 'foobarbaz')`
  - `showMatches('(a)\w+(b.z)', 'foobarbaz')`