Using Regular Expressions – with Kevin Skoglund (Lynda.com)

Table of Contents

[Chapter-0. Introduction](#h.30j0zll)

[File-01. 85870\_00\_00\_MM03\_introduction.mp4](#h.1fob9te)

[File-02. 85870\_00\_01\_SC11\_exercisefiles2.mp4](#h.3znysh7)

[Chapter-1. Regular Expressions](#h.2et92p0)

[File-01: 85870\_01\_01\_SC11\_whatareregex2.mp4](#h.tyjcwt)

[File-03: 85870\_01\_03\_SC11\_regexengines.mp4](#h.3dy6vkm)

[File-05: 85870\_01\_05\_SC11\_notation.mp4](#h.1t3h5sf)

[Chapter-2. Characters](#h.4d34og8)

[File-01: 85870\_02\_01\_SC11\_literalcharacters.mp4](#h.2s8eyo1)

[File-02: 85870\_02\_02\_SC11\_metacharacters.mp4](#h.17dp8vu)

[File-03: 85870\_02\_03\_SC11\_wildcard.mp4](#h.3rdcrjn)

[File-04: 85870\_02\_04\_SC11\_escaping.mp4](#h.26in1rg)

[File-05: 85870\_02\_05\_SC11\_specialchars.mp4](#h.lnxbz9)

[Chapter-3. Character Sets](#h.35nkun2)

[File-01: 85870\_03\_01\_SC11\_definecharset.mp4](#h.1ksv4uv)

[File-02: 85870\_03\_02\_SC11\_characterrange.mp4](#h.44sinio)

[File-03: 85870\_03\_03\_SC11\_negativecharset.mp4](#h.2jxsxqh)

[File-04: 85870\_03\_04\_SC11\_metacharinsets.mp4](#h.z337ya)

[File-05: 85870\_03\_05\_SC11\_shorthand2.mp4](#h.3j2qqm3)

[File-06: 85870\_03\_06\_SC11\_posixbrackets.mp4](#h.1y810tw)

[Chapter-4. Repetition Expressions](#h.4i7ojhp)

[File-01: 85870\_04\_01\_SC11\_repetitionmetachar.mp4](#h.2xcytpi)

[File-02: 85870\_04\_02\_SC11\_quantifiedrepetition.mp4](#h.1ci93xb)

[File-03: 85870\_04\_03\_SC11\_greedy.mp4](#h.3whwml4)

[File-04: 85870\_04\_04\_SC11\_lazy.mp4](#h.2bn6wsx)

[File-05: 85870\_04\_05\_SC11\_efficiency.mp4](#h.qsh70q)

[Chapter-5. Grouping and Alternation Expressions](#h.3as4poj)

[85870\_05\_01\_SC11\_grouping.mp4](#h.1pxezwc)

[85870\_05\_02\_SC11\_alternation.mp4](#h.49x2ik5)

[85870\_05\_03\_SC11\_logicalalternations.mp4](#h.2p2csry)

[85870\_05\_04\_SC11\_repeatandnest.mp4](#h.147n2zr)

[Chapter-6. Anchored Expressions](#h.3o7alnk)

[85870\_06\_01\_SC11\_startendanchors.mp4](#h.23ckvvd)

[85870\_06\_02\_SC11\_multilinemode.mp4](#h.ihv636)

[85870\_06\_03\_SC11\_wordboundaries.mp4](#h.32hioqz)

[Chapter-7. Capturing Groups and Backreferences](#h.1hmsyys)

[85870\_07\_01\_SC11\_backreferences.mp4](#h.41mghml)

[85870\_07\_02\_SC11\_backrefoptional.mp4](#h.2grqrue)

[85870\_07\_03\_SC11\_findreplace2.mp4](#h.vx1227)

[85870\_07\_04\_SC11\_noncapturing.mp4](#h.3fwokq0)

[Chapter-8. Lookaround Assertions](#h.1v1yuxt)

[85870\_08\_01\_SC11\_positivelookahead.mp4](#h.4f1mdlm)

[85870\_08\_02\_SC11\_doubletesting.mp4](#h.2u6wntf)

[85870\_08\_03\_SC11\_negativelookahead.mp4](#h.19c6y18)

[85870\_08\_04\_SC11\_lookbehind.mp4](#h.3tbugp1)

[85870\_08\_05\_SC11\_powerpositions.mp4](#h.28h4qwu)

[Chapter-9. Unicode and Multibyte Characters](#h.nmf14n)

[85870\_02\_06a\_SC11\_unicode2.mp4](#h.37m2jsg)

[85870\_02\_06b\_SC11\_unicode2.mp4](#h.1mrcu09)

[85870\_02\_07\_SC11\_unicodewildcard.mp4](#h.46r0co2)

[Chapter-10. Useful Regular Expressionsss](#h.2lwamvv)

[85870\_09\_01\_SC11\_howtouse.mp4](#h.111kx3o)

[85870\_09\_02\_SC11\_matchnames.mp4s](#h.3l18frh)

[85870\_09\_03\_SC11\_matchpostal.mp4](#h.206ipza)

[85870\_09\_04\_SC11\_matchemails.mp4](#h.4k668n3)

[85870\_09\_05\_SC11\_matchurls2.mp4](#h.2zbgiuw)

[85870\_09\_06\_SC11\_decimal2.mp4](#h.1egqt2p)

[85870\_09\_07\_SC11\_ipaddress.mp4](#h.3ygebqi)

[85870\_09\_08\_SC11\_matchdates.mp4](#h.2dlolyb)

[85870\_09\_09\_SC11\_matchtimes.mp4](#h.sqyw64)

[85870\_09\_10\_SC11\_matchtags.mp4](#h.3cqmetx)

[85870\_09\_11\_SC11\_matchpasswords3.mp4](#h.1rvwp1q)

[85870\_09\_12\_SC11\_matchcreditcards.mp4](#h.4bvk7pj)

[85870\_09\_13\_SC11\_wordsnearwords.mp4](#h.2r0uhxc)

[85870\_09\_14\_SC11\_formatfirst.mp4](#h.1664s55)

[85870\_09\_15\_SC11\_formatsecond.mp4](#h.3q5sasy)

[85870\_09\_16\_SC11\_formatthird.mp4](#h.25b2l0r)

[Chapter-11. Conclusion](#h.kgcv8k)

[85870\_10\_01\_SC11\_conclusion.mp4](#h.34g0dwd)

# **Chapter-0. Introduction**

## **File-01. 85870\_00\_00\_MM03\_introduction.mp4**

## **File-02. 85870\_00\_01\_SC11\_exercisefiles2.mp4**

# **Chapter-1. Regular Expressions**

## **File-01: 85870\_01\_01\_SC11\_whatareregex2.mp4**

Description:  
1. What are Regular Expressions?  
a) Regular Expression  
 - Symbols representing a text pattern.  
b) Regular Expressions  
 - Formal language interpreted by a regular expression processor.  
c) Used for matching, searching and replacing text.  
d) “Regex” for short (sometimes “Regexp”)  
e) Usage examples  
 - Test if a phone number has the correct number of digits.  
 - Test if an email address is in a valid format.  
 - Search a document for either “color” or “colour”.  
 - Replace all occurrences of “Bob”, “Bobby”, or “B.” with “Robert”  
 - Count the number of times “training” is immediately preceded by “computer”, “video, or “online”  
 - Convert a tab-delimited file into a comma-delimited file  
 - Find duplicate words in a text.  
f) “matches”  
 - A regular expression matches text if it correctly describes the text  
 - Text matches a regular expression if it is correctly described by the expression

File-02: 85870\_01\_02\_SC11\_history.mp4

Description:  
1. History of Regular Expressions  
a) 1943: Warren McCulloch and Walter Pitts develop models of how the nervous system works (i.e., how a machine could be built like a brain)  
b) 1956: Stephen Kleene describes these models with an algebra called “regular sets” and creates a notation to express them called “regular expressions”  
c) 1968: Ken Thompson implements regular expressions in ed, an early Unix text editor  
d) g/Regular Expression/p  
e) Global Regular Expression Print: “grep”  
f) grep become stand-alone program  
g) Became widely used in other Unix programs (awk, vi, emacs, et al.)  
h) Popularity of Unix and grep  
i) grep evolves   
j) egrep: “extended grep”  
k) Many programs, many programmers, many changes = many imcompatibilities  
l) 1986: POSIX (Portable Operating System Interface)  
 - Standard to ensure compatibility  
 - Basic Regular Expressions (BREs)  
 - Extended Regular Expressions (EREs)  
m) 1986: Henry Spencer releases a regex library written in C  
 - Could be incorporated into other programs  
 - Provided consistency  
n) 1987: Larry Wall releases Perl  
 - Used Spencer's regex library  
 - Over time, added many powerful features  
o) Perl-compatible languages and programs  
 - Apache, C/C++, C#/VB/.NET, Java, JavaScript, MySQL, PHP, Python, Ruby  
 - PCRE library

## **File-03: 85870\_01\_03\_SC11\_regexengines.mp4**

Description:  
1. Regular Expression Engines  
 - C/C++  
 - Java  
 - JavaScript/ActionScript (ECMAScript)  
 - .NET  
 - Perl  
 - PHP (PCRE)  
 - Python  
 - Ruby  
 - Unix (POSIX BRE, POSIX ERE)  
 - Apache (v1: POSIX ERE, v2: PCRE)  
 -MySQL (POSIX ERE)  
File-04: 85870\_01\_04\_SC11\_installeditor.mp4Description:  
1. Installing An Engine  
 - grep/egrep (old)  
 - TextMate/e Text Editor  
 - BBEdit  
 - EditPad Pro  
 - PowerGREP  
 - RegexBuddy  
 - RegexMagic  
 - C/C++, Java, .NET, Perl, PHP, Python, Ruby  
 - Mobile applications  
 - Online, JavaScript version  
 - [http://regexpal.com](http://regexpal.com/)  
 - Included with the course files

## **File-05: 85870\_01\_05\_SC11\_notation.mp4**

Description:  
1. Notation Conventions and Modes  
a) Regular expression  
 - /abc/  
 - As in: g/re/p  
 - Use without forward slashes  
b) Text string  
 - “abc”   
 - Use without quotes  
c) Modes  
 - Standard: /re/  
 - Global: /re/g  
 - Case-insensitive: /re/i  
 - Multiline: /re/m  
 - Dot-matches-all: /re/s

# **Chapter-2. Characters**

## **File-01: 85870\_02\_01\_SC11\_literalcharacters.mp4**

Description:  
1. Strings  
a) /car/ matches “car”  
b) /car/ matches the first three letters of “carnival”  
c) Similar to searching in a word processor  
d) Simplest match there is  
2. Case-sensitive (by default)  
3. Standard (non-global) matching  
a) Earliest (leftmost) match is always preferred  
b) /zz/ matches the first set of z's in “pizzazz”  
4. Global matching  
a) All matches are found throughout the text  
b) /zz/g matches the both sets of z's in “pizzazz”  
5. Regular expression engines are eager.  
6.   
Regex:  /car/  
String: "car"  
  
Regex:  /car/  
String: "carnival"  
  
  
Case-sensitivity matters  
Regex:  /car/  
String: "Carnival"  
  
Regex:  /car/i  
String: "Carnival"  
  
Regex:  /Car/i  
String: "carnival"  
  
  
Whitespace matters  
Regex:  /car/  
String: "c a r"  
  
  
Global modifier  
Regex:  /zz/  
String: "pizzazz"  
  
Regex:  /zz/g  
String: "pizzazz"  
  
Regex:  /cat/  
String: "The cow, camel, and cat communicated."  
  
Regex:  /cat/g  
String: "The cow, camel, and cat communicated."

## **File-02: 85870\_02\_02\_SC11\_metacharacters.mp4**

Description:  
Heading: Metacharacters  
1. Characters with special meaning  
a) Like mathematical operators  
b) Transform literal characters into powerful expressions  
2. Only a few metacharacters to learn  
a) \ . \* + - {} [] ^ $ | ? () : ! =  
3. Can have more than one meaning  
a) Depends on how it is used in context  
4. Variation between regex engines  
5.   
Notice the syntax coloring  
  
Regex: . \* + [abc] ^ $ | ? (abc) \d{2,3}

## **File-03: 85870\_02\_03\_SC11\_wildcard.mp4**

Description:  
Heading: The Wildcard Metacharacter  
1. Metacharacter: .  
 Meaning: Any character except newline  
2. Matches any one character except newline  
a) Original Unix regex tools were line-based  
b) /h.t/ matches “hat”, “hot”, and “hit”, but not “heat”  
3. Broadest match possible  
4. Most common metacharacter  
5. Most common mistake  
a) /9.00/ matches “9.00”, “9500”, and “9-00”  
6. The Challenge of regular expressions is both matching what you want and only what you want  
7.   
Regex:  /h.t/  
String: "hat hot hit heat hzt h t h#t h:t"  
  
Regex:  /.a.a.a/  
String: "banana papaya abacab"  
  
Regex:  /a.a.a./  
String: "banana papaya abacab"  
  
Write a regex that matches all three words:  
String: "silver sliver slider"

## **File-04: 85870\_02\_04\_SC11\_escaping.mp4**

Description:  
1. Metacharacter: \  
 Meaning: Escape the next character  
2. Allows use of metacharacters as literal characters  
a) Match a period with \.  
b) /9\.00/ matches “9.00”, but not “9500” or “9-00”  
c) Match a backslash by escaping a backslash (\\)  
3. Only for metacharacters  
a) Literal characters should never be escaped, gives them meaning  
4. Quotation marks are not metacharacters, do not need to be escaped  
5.   
Regex:  /9\.00/  
String: "9.00 9500 9-00"  
  
String: "his\_export.txt her\_export.txt"  
Regex:  /h..\_export.txt/  
Regex:  /h..\_export\.txt/  
  
String: "resume1.txt resume2.txt resume3\_txt.zip"  
Regex:  /resume..txt/  
Regex:  /resume.\.txt/  
  
  
Also may need to escape /  
  
String: "/home/user/document.txt"  
Regex:  //home/user/document\.txt/  
Regex:  /\/home\/user\/document\.txt/

## **File-05: 85870\_02\_05\_SC11\_specialchars.mp4**

Description:  
Heading: Other Special Characters  
1. Spaces  
2. Tabs (\t)  
3. Line returns (\r, \n, \r\n)  
4. Non-printable characters  
a) bell (\a), escape (\e), form feed (\f), vertical tab (\v)  
5. ASCII or ANSI codes  
a) Codes that control appearance of a text terminal  
b) 0xA9 = \xA9  
6.   
Regex:  /c a t/  
String: "c a t"  
  
Regex:  /a\tb/  
String: "a  b"  
  
Regex:  /c\nd/  
String: "abc  
def"

# **Chapter-3. Character Sets**

## **File-01: 85870\_03\_01\_SC11\_definecharset.mp4**

Description:  
Heading: Defining a Character Set  
1. Metacharacter: [  
 Meaning: Begin a character set  
 Metacharacter: ]  
 Meaning: End a character set  
2. Any one of several characters  
a) But only one character  
b) Order of chareacters does not matter  
3. Examples  
a) /[aeiou]/ matches any one vowel  
b) /gr[ea]y/ matches “grey” and “gray”  
c) /gr[ea]t/ does not match “great”  
4.   
Matches single, lowercase vowels  
Regex:  /[aeiou]/  
String: "Bananas Peaches Apples"  
  
Regex: /gr[ea]y/  
String: "grey gray"  
  
Regex: /gr[ea]t/  
String: "great"  
  
Regex: /gr[ea][ea]t/  
String: "great graet greet graat"

## **File-02: 85870\_03\_02\_SC11\_characterrange.mp4**

Description:  
Heading: Character Ranges  
1. Metacharacter: -  
 Meaning: Range of characters  
2. Range metacharacter  
a) Represents all characters between two characters  
b) Only a metacharacter inside a character set, a literal dash otherwise  
3. Examples  
a) [0-9]  
b) [A-Za-z]  
c) [a-ek-ou-y]  
4. Caution  
a) [50-99] is not all numbers from 50 to 99, it is the same as [0-9]  
5.   
Regex: /[ABCDEFGHIJKLMNOPQRSTUVWXYZ]ello/  
String: "Hello"  
  
Regex: /[A-Z]ello/  
String: "Hello"  
  
Regex: /[A-Za-z]/  
String: "Hello"  
  
  
Phone number  
  
String: "555-666-7890"  
Regex: /[0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]/  
  
  
Postal codes  
  
String: "90210"  
Regex:  /[0-9][0-9][0-9][0-9][0-9]/  
  
String: "WC2H 9AW"  
Regex:  "[0-9A-Z][0-9A-Z][0-9A-Z][0-9A-Z] [0-9A-Z][0-9A-Z][0-9A-Z]"

## **File-03: 85870\_03\_03\_SC11\_negativecharset.mp4**

Description:  
Heading: Negative Character Sets  
1. Metacharacter: ^  
 Meaning: Negate a character set  
2. Not any one of several characters  
a) Add ^ as the first character inside a character set  
b) Still represents one character  
3. Examples  
a) /[^aeiou]/ matches any one consonant (non-vowel)  
b) /see[^mn]/ matches “seek” and “sees” but not “seem” or “seen”  
3. Caution  
a) /see[^mn]/ matches “see ” but not “see”  
4.   
Regex:  /[^a-z]/  
String: "Now we know how to make negative character sets."  
  
  
Negates the entire character set  
  
Regex:  /[^a-zA-Z]/  
String: "Now we know how to make negative character sets."  
  
  
Only matches a single character  
  
Regex:  /[^aeiou]/  
String: "It seems I see the sea I seek."  
  
Regex: /see[^mn]/  
String: "It seems I see the sea I seek."  
  
  
Must match one character (which may be a space or punctuation)  
  
Regex:  /see[^mn]/  
String: "It seems I see the sea I see"  
  
Regex:  /see[^mn]/  
String: "It seems I see the sea I see."

## **File-04: 85870\_03\_04\_SC11\_metacharinsets.mp4**

Description:  
Heading: Metacharacters inside Character Sets  
1. Metacharacters inside character sets are already escaped  
a) Do not need to escape them again  
b) /h[abc.xyz]t/ matches “hat” and “h.t”, but not “hot”  
2. Exceptions  
a) ] - ^ \  
3. Examples  
a) /var[[(][0-9][\])]/  
b) /2003[-/]10[-/]05/ may not require escaping  
c) /file[0-\\_]1/ does require escaping  
4.   
Regex:  /h[abc.xyz]t/  
String: "hat hot h.t"  
  
Regex:  /var[[(]\d[\])]/  
String: "var(3) var[4]"  
  
May not require escaping  
Regex:  /2003[-/]10[-/]05/  
String: "2003/10/05 2003-10-05"  
  
Regex:  /file[0-\\_]1/  
String: "file01 file-1 file\1 file\_1"

## **File-05: 85870\_03\_05\_SC11\_shorthand2.mp4**

Description:  
Heading: Shorthand Character Sets  
1. Shorthand → Meaning → Equivalent  
 \d → Digit → [0-9]  
 \w → Word character → [a-zA-Z0-9]  
 \s → Whitespace → [ \t\r\n]  
 \D → Not digit → [^0-9]  
 \W → Not word character → [^a-zA-Z0-9\_]  
 \S → Not whitespace → [^ \t\r\n]  
2. \w  
a) Underscore is a word character  
b) Hyphen is not a word character  
3. Examples  
a) /\d\d\d\d/ matches “1984”, but not “text”  
b) /\w\w\w/ matches “ABC”, “123”, and “1\_A”  
c) /\w\s\w\w/ matches “I am”, but not “Am I”  
d) /[\w\-]/ matches as word character or hyphen (useful)  
e) /[\d\s]/ matches any digit or whitespace character  
f) /[^\d]/ is the same as /\D/ and /[^0-9]/  
4. Caution  
a) /[^\d\s]/ is not the same as [\D\S]  
b) /[^\d\s]/ = NOT digit OR space character  
c) /[\D\S]/ = EITHER NOT digit OR NOT space character  
5. Support  
a) Originated with Perl  
b) All modern regex engines  
c) Not in many Unix tools  
6.   
Regex:  /\d\d\d\d/  
String: "1984 text"  
  
Regex:  /\w\w\w\w/  
String: "1984 text 1\_5W"  
  
Regex:  /[\w\-]/  
String: "blue-green paint"  
  
Regex:  /[\d\s]/  
String: "123 456 789 abc"  
  
  
Be careful when using negatives  
  
Regex:  /[^\d\s]/  
String: "123 456 789 abc"  
  
Regex:  /[\D\S]/  
String: "123 456 789 abc"

## **File-06: 85870\_03\_06\_SC11\_posixbrackets.mp4**

Description:  
Heading: POSIX Bracket Expressions  
1. Class → Meaning → Equivalent  
 [:alpha:] → Alphabetic characters → A-Za-z  
 [:digit:] → Numeric characters → 0-9  
 [:alnum:] → Alphanumeric characters → A-Za-z0-9  
 [:lower:] → Lowercase alphabetic characters → a-z  
 [:upper:] → Uppercase alphabetic characters → A-Z  
 [:punct:] → Punctuation characters  
 [:space:] → Space characters  
 [:blank:] → Blank characters (space, tab)  
 [:print:] → Printable characters, spaces  
 [:graph:] → Printable characters, no spaces  
 [:cntrl:] → Control characters (non-printable)  
 [:xdigit:] → Hexadecimal characters → A-Fa-f0-9  
2. Use inside a character class, not standalone  
a) Correct: [[:alpha:]] or [^[:alpha:]]  
b) Incorrect: [:alpha:]  
3. Good idea not to mix POSIX sets and other shorthand sets  
4. Support  
a) Yes: Perl, PHP, Ruby, Unix  
b) No: Java, JavaScript, .NET, Python  
5.   
This command only works on Unix  
  
ps aux | grep --regexp="s[[:digit:]]"

# **Chapter-4. Repetition Expressions**

## **File-01: 85870\_04\_01\_SC11\_repetitionmetachar.mp4**

Description:  
Heading: Repetition Metacharacters  
1.

|  |  |
| --- | --- |
| Metacharacter | Meaning |
| \* | Preceding item zero or more times |
| + | Preceding item one or more times |
| ? | Preceding item zero or one time |

2. Examples  
a) /apples\*/ matches “apple”, “apples”, and “applesssssssss”  
b) /apples+/ matches “apples” and “applesssssss”, but not “apple”  
c) /apples?/ matches “apple” and “apples”, but not “applessssssss”  
d) /\d\d\d\d\*/ matches numbers with three digits or more  
e) /\d\d\d+/ matches numbers with three digits or more  
f) /colou?r/ matches “color” and “colour”  
3. Support  
a) \* is supported in all regex engines  
b) + and ? are not supported in BREs (i.e., old Unix programs)  
4.   
Regex:  /apples\*/  
String: "apple apples applesssssss"  
  
Regex:  /apples+/  
String: "apple apples applesssssss"  
  
Regex:  /apples?/  
String: "apple apples applesssssss"  
  
  
Regex:  /\d\d\d\d\*/  
String: "123456789 1234 123 12"  
  
Regex:  /\d\d\d+/  
String: "123456789 1234 123 12"  
  
  
Regex:  /[a-z]+\d[a-z]\*/  
String: "abc9xyz"  
String: "a9xyz"  
String: "9xyz"  
  
Regex:  /[a-z]+\d[a-z]\*/  
String: "abc9xyz"  
String: "abc9z"  
String: "abc9"  
  
  
Optional letter "u"  
  
Regex:  /colou?r/  
String: "color colour"  
  
  
Find any word that ends in "s"  
  
Regex:  /\w+s/  
String: "We picked apples."

## **File-02: 85870\_04\_02\_SC11\_quantifiedrepetition.mp4**

Description:  
Heading: Quantified Repetition  
1.

|  |  |
| --- | --- |
| Metacharacter | Meaning |
| { | Start quantified repetition of preceding item |
| } | End quantified repretition or preceding item |

2. {min, max}  
a) min and max are positive numbers  
b) min must always be included, can be zero  
c) max is optional  
3. Three syntaxes  
a) \d{4,8} matches numbers with four to eight digits  
b) \d{4} matches numbers with exactly four digits (min is max)  
c) \d{4, } matches numbers with four or more digits (max is infinite)  
4. Examples  
a) \d{0, } is the same as \d\*  
b) \d{1, } is the same as \d+  
c) /\d{3}-\d{3}-\d{4}/ matches most U.S. phone numbers  
d) /A{1,2} bonds/ matches “A bonds” and “AA bonds”, not “AAA bonds”  
5.   
Regex: /\w{5}\s/  
String: (use shakespeare\_sonnet.txt)  
  
Regex: /\w{2,5}\s/  
Regex: /\w{5,}\s/  
  
  
Regex: /\d{3}-\d{3}-\d{4}/  
String: 555-867-5309  
  
Regex: /A{1,2} bonds/  
String: "A bonds AA bonds AAA bonds"  
  
  
Regex: /\w+\_\d+-\d+/  
String: "report\_1997-04 budget\_03-04 memo\_712539-100"  
  
Regex: /\w+\_\d{2,4}-\d{2}/  
String: "report\_1997-04 budget\_03-04 memo\_712539-100"

shakespeare\_sonnet:  
Shall I compare thee to a summer's day?  
Thou art more lovely and more temperate:  
Rough winds do shake the darling buds of May,  
And summer's lease hath all too short a date.

## **File-03: 85870\_04\_03\_SC11\_greedy.mp4**

Description:  
Heading: Greedy Expressions  
01. Example 1  
a) 01\_FY\_07\_report\_99.xls  
b) /\d+\w+\d+/  
02. Example 2  
a) “Milton”, “Waddams”, “Initech, Inc.”  
b)) /“.+”, “.+”/  
03. Standard repetition quantifiers are greedy  
04. Expression tries to match the longest possible string  
05. Defers to achieving overall match  
a) /.+\.jpg/ matches “filename.jpg”  
b) The + is greedy, but “gives back” the “.jpg” to take match  
c) Think of it as rewinding or backtracking  
06. Gives back as little as possible  
a) /.\*[0-9]+/ matches “Page 266”  
b) /.\*/ matches “Page 26” while /[0-9]+/ matches “6”  
07. Regular expression engines are eager  
08. Regular expression engines are greedy  
09.  
  
Regex: /.+\.jpg/  
String: "filename.jpg"  
  
Regex: /.\*[0-9]+/  
String: "Page 266"  
  
Regex: /\d+\w+\d+/  
String: "01\_FY\_07\_report\_99.xls"  
  
Regex: /".+", ".+"/  
String: "Milton", "Waddams", "Initech, Inc."

## **File-04: 85870\_04\_04\_SC11\_lazy.mp4**

Description:  
Heading: Lazy Expressions  
01.

|  |  |
| --- | --- |
| Metacharacter | Meaning |
| ? | Make preceding quantifier lazy |

02. Syntax  
a) \*?  
b) +?  
c) {min, max}?  
d) ??  
03. Instructs quantifier to use a “lazy strategy” for making choices  
04. Greedy strategy  
a) Match as much as possible before giving control to the next expression part  
05. Lazy strategy  
a) Match as little as possible before giving control to the next expression part  
b) Still defers to overall match  
c) Not necessarily faster or slower  
06. Examples  
a) /\w\*?\d{3}/  
b) /[A-Za-z-]+?\./  
c) /.{4,8}?\_.{4,8}/  
d) /apples??/  
07. Support  
a) Not supported in most Unix tools (BRE, ERE)  
08.   
Regex: /.\*?[0-9]+/   
String: Page 266  
  
Regex: /.\*?[0-9]\*?/  
String: Page 266  
09. Example 1  
a) 01\_FY\_07\_report\_99.xls  
b) /\d+\w+?\d+/  
10. Example 2  
a) “Milton”, “Waddams” “Initech, Inc.”  
b) /“.+?”, “.+?”/  
11.   
Remove the ? from each regex and watch the changes.  
  
Regex: /\w\*?\d{3}/  
String: "image\_294"  
  
Regex: /[A-Za-z-.]+?\./  
String: "Dr. Roberts, M.D."  
  
Regex: /.{4,8}\_.{2,6}?/  
String: "last\_qtr\_report.xls"  
  
Regex: /apples??/  
String: "We picked apples."

## **File-05: 85870\_04\_05\_SC11\_efficiency.mp4**

Description:  
Heading: Efficiency when using repetition  
01. Efficient matching + less backtracking = speedy results  
02. Define the quantity of repeated expressions  
a) /.+/ is faster than /.\*/  
b) /.{5}/ and /.{3,7}/ are even faster  
03. Narrow the scope of the repeated expression  
a) /.+/ can become /[A-Za-z]+/  
04. Provide clearer starting and ending points  
a) /<.+>/ can become /<[^>]+>/  
b) Use anchors and word boundaries  
05. Example   
a) /\w\*s/ would be improved as /\w+s/  
b) /\w+s/ would be improved as /[A-Za-z]+s/  
c) Perhaps as /[a-z]+s/ or as /[A-Z][a-z]+s/  
d) Search for whole words only.  
e) Spaces, anchors, or word boundaries  
f) Scans “picked” but not “icked”, “cked”, “ked”, “ed”, or “d”  
06.  
Regex: /\w\*s/  
String: "We picked apples."  
  
Regex: /\w+s/  
String: "We picked apples."  
  
Regex: /[A-Za-z]+s/  
String: "We picked apples."  
  
Regex: /\s[A-Za-z]+s/  
String: "We picked apples."  
  
Regex: /\s[a-z]+s/  
String: "We picked apples."  
  
Regex: /\s[a-z]{5}s/  
String: "We picked apples."

# **Chapter-5. Grouping and Alternation Expressions**

## **File-01: 85870\_05\_01\_SC11\_grouping.mp4**

Description:  
Heading: Grouping Metacharacters  
01.

|  |  |
| --- | --- |
| Metacharacter | Meaning |
| ( | Start grouped expression |
| ) | End grouped expression |

02. Group portions of the expression  
a) Apply repetition operators to a group  
b) Makes expressions easier to read  
c) Captures group for use in matching and replacing  
d) Cannot be used inside character set  
03. Examples  
a) /(abc)+/ matches “abc” and “abcabcabc”  
b) /(in)?dependent/ matches “independent” and “dependent”  
c) /run(s)?/ is the same as /runs?/  
04.   
Regex: /[A-Z][0-9]/  
String: "A1B2C3D4E5F6G7H8I9J0"  
  
Regex: /([A-Z][0-9])/  
String: "A1B2C3D4E5F6G7H8I9J0"  
  
Regex: /([A-Z][0-9])+/  
String: "A1B2C3D4E5F6G7H8I9J0"  
  
Regex: /([A-Z][0-9]){3}/  
String: "A1B2C3D4E5F6G7H8I9J0"  
  
Regex: /(in)dependent/  
String: "dependent or independent"  
  
Regex: /(in)?dependent/  
String: "dependent or independent"  
  
Regex: /runs?/  
String: "I run fast. He runs faster."  
  
Regex: /run(s)?/  
String: "I run fast. He runs faster."

## **File-02: 85870\_05\_02\_SC11\_alternation.mp4**

Description:  
Heading: Alternation Metacharacter  
01.

|  |  |
| --- | --- |
| Metacharacter | Meaning |
| | | Match previous or next expression |

02. | is an OR operator  
a) Either match expression on the left or match expression on the right  
b) Ordered, leftmost expression gets precedence  
c) Multiple choices can be daisy-chained  
d) Group alternation expressions to keep them distinct  
03. Examples  
a) /apple|orange/ matches “apple” and “orange”  
b) /abc|def|ghi|jkl/ matches “abc”, “def”, “ghi” and “jkl”  
c) /apple(juice|sauce)/ is not the same as /applejuice|sauce/  
d) /w(ei|ie)rd/ matches “weird” and “weird”  
04.   
Regex: /apple|orange/  
String: "apple orange appleorange apple|orange"  
  
Regex: /abc|def|ghi|jkl/  
String: "abcdefghijklmnopqrstuvwxyz"  
  
  
Grouping is always a good idea, and sometimes required.  
  
Regex: /applejuice|sauce/  
String: "applejuice applesauce"  
  
Regex: /apple(juice|sauce)/  
String: "applejuice applesauce"  
  
  
Find misspelled words.  
  
Regex: /w(ei|ie)rd/  
String: "weird and wierd"

## **File-03: 85870\_05\_03\_SC11\_logicalalternations.mp4**

Description:  
Heading: Writing logical and efficient alternations  
01. Regular expression engines are eager.  
02. Regular expression engines are greedy.  
03. Put simplest (most efficient) expression first [number (b) is most efficient]  
a) /\w+\_\d{2,4}|\d{4}\_\d{2}\_\w+|export\d{2}/  
b) /export\d{2}|\d{4}\_\d{2}\_\w+|\w+\_\d{2,4}/  
04.   
Regex: (peanut|peanutbutter)  
String: "peanutbutter"  
  
Regex: peanut(butter)?  
String: "peanutbutter"  
  
Regex: (\w+|FY\d{4}\_report.xls)  
String: "FY2003\_report.xls"  
  
  
Turn off global matching and notice which gets matched.  
  
Regex: /abc|def|ghi|jkl/  
String: "abcdefghijklmnopqrstuvwxyz"  
  
Regex: /xyz|abc|def|ghi|jkl/  
String: "abcdefghijklmnopqrstuvwxyz"  
  
  
Which is more efficient?  
  
Regex: /\w+\_\d{2,4}|\d{4}\_\d{2}\_\w+|export\d{2}/  
  
Regex: /export\d{2}|\d{4}\_\d{2}\_\w+|\w+\_\d{2,4}/

## **File-04: 85870\_05\_04\_SC11\_repeatandnest.mp4**

Description:  
Heading: Repeating and nesting alternations  
01. Repeating  
a) First matched alternation does not effect the next matches  
b) /(AA|BB|CC){6}/ matches “AABBAACCAABB”  
02. Nesting  
a) Check nesting carefully  
b) /(\d{2}([A-Z]{2}|-\d\w\d\w)|\d{4}(-\d{2}-[A-Z]{2,8}|\_x[A-F]))/  
c) Trade-off between precision, readability, and efficiency  
03.   
Repetition  
  
Regex: /(\d\d|[A-Z][A-Z]){3}/  
String: "112233 AABBCC AA66ZZ 11AA44"  
  
  
Nesting  
  
Regex: /(apple (juice|sauce)|milk(shake)?|sweet (peas|corn|potatoes))/  
String: (use shopping\_list.txt)  
  
Regex: /(apple juice|apple sauce|milk|milk shake|sweet peas|sweet corn|sweet potatoes)/  
String: (use shopping\_list.txt)  
  
Regex: /[\w ]+/  
String: (use shopping\_list.txt)  
04. Shopping\_list.txt is:  
milk  
apple juice  
sweet peas  
yogurt  
sweet corn  
apple sauce  
milkshake  
sweet potatoes

# **Chapter-6. Anchored Expressions**

## **File-01: 85870\_06\_01\_SC11\_startendanchors.mp4**

Description:  
Heading: Start and End Anchors  
01.

|  |  |
| --- | --- |
| Metacharacter | Meaning |
| ^ | Start of string/line |
| $ | End of string/line |
| \A | Start of string, never end of line |
| \Z | End of string, never end of line |

02. Reference a position, not an actual character  
a) Zero-width  
03. Examples  
a) /^apple/ or /\Aapple/  
b) /apple$/ or /apple\Z/  
c) /^apple$/ or /\Aapple\Z/  
04. Support  
a) ^ and $ are supported in all regex engines  
b) \A and \Z are supported in Java, .NET, Perl, PHP, Python, Ruby  
05.   
Regex: /[A-Z]/  
String: "Mr. Smith went to Washington."  
  
Regex: /^[A-Z]/  
String: "Mr. Smith went to Washington."  
  
Regex: /\./  
String: "Mr. Smith went to Washington."  
  
Regex: /\.$/  
String: "Mr. Smith went to Washington."  
  
Regex: /^[A-Z][A-Za-z.\- ]+\.$/  
String: "Mr. Smith went to Washington."  
  
  
Regex: /^\w+@\w+\.[a-z]{3}$/  
String: "nobody@nowhere.com"  
  
Regex: /^\w+@\w+\.[a-z]{3}$/  
String: "nobody@nowhere.com, somebody@somewhere.com"  
  
  
Find whitespace  
  
Regex: /^[ \t]+/  
String: " It was a dark and stormy night."  
  
Regex: /[ \t]+$/  
String: "And they lived happily ever after. "

## **File-02: 85870\_06\_02\_SC11\_multilinemode.mp4**

Description:  
Heading: Line breaks and multiline mode  
01.

|  |  |
| --- | --- |
| Metacharacter | Meaning |
| ^ | Start of string/line |
| $ | End of string/line |
| \A | Start of string, never end of line |
| \Z | End of string, never end of line |

02. Single-line mode  
a) ^ and $ do not match at line breaks  
b) \A and \Z do not match at line breaks  
c) Many Unix tools support only single line  
03. Multiline mode  
a) ^ and $ will match at start and end of lines  
b) \A and \Z do not match at line breaks  
c) Languages usually offer a multiline option  
04. Multiline mode in different languages  
a) Java: Pattern.compile(“^regex$”, Pattern.MULTILINE)  
b) JavaScript: /^regex$/m  
c) .NET: Regex.Match(“string”, “^regex$”, RegexOptions.Multiline)  
d) Perl: m/^regex$/m  
e) PHP: preg\_match(/^regex$/m, “string”)  
f) Python: re.search(“^regex$”, “string”, re.MULTILINE)  
g) Ruby: string.match(/^regex$/m)  
05.   
Regex: /^[a-z ]+/  
String: (see shopping\_list.txt)  
  
Regex: /[a-z ]+$/  
String: (see shopping\_list.txt)  
  
  
Using Multiline mode  
  
Regex: /^[a-z ]+/m  
String: (see shopping\_list.txt)  
  
Regex: /[a-z ]+$/m  
String: (see shopping\_list.txt)  
06.  
milk  
apple juice  
sweet peas  
yogurt  
sweet corn  
apple sauce  
milkshake  
sweet potatoes

## **File-03: 85870\_06\_03\_SC11\_wordboundaries.mp4**

Description:  
Heading: Word Boundaries  
01.

|  |  |
| --- | --- |
| Metacharacter | Meaning |
| \b | Word boundary (start/end of word) |
| \B | Not a word boundary |

02. Reference a position, not an actual character  
03. Conditions for matching  
a) Before the first word character in the string  
b) After the last word character in the string  
c) Between a word character and a non-word character  
04. Word characters: [A-Za-z0-9\_]  
05. Support  
a) Most regex engines, not in early Unix tools (BREs)  
06. Boundary examples  
a) /\b\w+\b/ finds four matches in “This is a test.”  
b) /\b\w+\b/ matches all of “abc\_123” but only part of “top-notch”  
07. Not a boundary examples  
a) /\BThis/ does not match “This is a test.”  
b) /\B\w+\B/ finds two matches in “This is a test.” (“hi” and “es”)  
08. Caution  
a) A space is not a word boundary  
b) Word boundaries reference a position  
 - Not an actual character  
 - Zero-length  
09. Examples  
a) String: “apples and oranges”  
b) No match: /apples\band\boranges/  
c) Match: /apples\b \band\b \boranges/  
10.  
Regex: /\b\w+\b/  
String: (see shakespeare\_sonnet.txt)  
  
Regex: /\b[\w']+\b/  
String: (see shakespeare\_sonnet.txt)  
  
Regex: /\b[\w']+?\b/  
String: (see shakespeare\_sonnet.txt)  
  
  
Faster matches using word boundaries  
  
Regex: /\w+s/  
String: "We picked apples."  
  
Regex: /\b\w+s\b/  
String: "We picked apples."  
11. Shakespeare\_sonnet.txt  
Shall I compare thee to a summer's day?  
Thou art more lovely and more temperate:  
Rough winds do shake the darling buds of May,  
And summer's lease hath all too short a date.

# **Chapter-7. Capturing Groups and Backreferences**

## **File-01: 85870\_07\_01\_SC11\_backreferences.mp4**

Description:  
Heading: Backreferences  
01. Grouped expressions are captured  
a) Stores the matched portion in parentheses  
b) /a(p{2}l)e/ matches “apple” and stores “ppl”   
c) Stores the data matched, not the expression  
d) Automatically, by default  
02. Backreferences allow access to captured data  
a) Refer to first backreference with \1  
03.

|  |  |
| --- | --- |
| Metacharacter | Meaning |
| \1 through \9 | Backreference for positions 1 to 9 |

04. Usage  
a) Can be used in the same expression as the group  
b) Can be accessed after the match is complete  
c) Cannot be used inside character classes  
05. Support  
a) Most regex engines support \1 through \9  
b) Some regex engines support \10 through \99  
c) Some regex engines use $1 through $9 instead  
06. Examples  
a) /(apples) to \1/ matches “apples to apples”  
b) /(ab)(cd)(ef)\3\2\1/ matches “abcdefefcdab”  
c) /<(i|em)>.+?</\1>/ matches “<i>Hello</i>” and “<em>Hello</em>  
 - Does not match “<i>Hello</em>” or “<em>Hello</i>”  
07.   
Regex: /(apples) to \1/  
String: "apples to apples"  
  
Regex: /(ab)(cd)(ef)\3\2\1/  
String: "abcdefefcdab"  
  
Regex: /(ab)(cd)(ef)(gh)(ij)\3\2\1\4\5/  
String: "abcdefghijefcdabghij"  
  
Regex: /<(i|em)>.+?</\1>/  
String: "<i>italics</i> <em>emphasis</em> <i>bad</em> <em>bad</i>"  
  
Regex: /<(i|em|b|strong)>.+</\1>/  
String: "<b>bold</b> <strong>strong</strong>"  
  
Regex: /\b([A-Z][a-z]+)\b\s\b\1son\b/  
String: "Steve Smith, John Johnson, Eric Erikson, Evan Evanson"  
  
Regex: /\b(\w+)\s+\1\b/  
String: "Paris in the   
 the spring."

## **File-02: 85870\_07\_02\_SC11\_backrefoptional.mp4**

Description:  
Heading: Backreferences to optional expressions  
01. Optional elements  
a) /A?B/ matches “AB” and “B”  
02. Captures occur on zero-width matches  
a) /(A?)B/ matches “AB” and captures “A”  
b) /(A?)B/ matches “B” and captures “”  
03. Backreferences become zero-width too  
a) /(A?)B\1/ matches “ABA” and “B”  
b) /(A?)B\1C/ matches “ABAC” and “BC”  
04. Optional elements  
a) /A?B/ matches “AB” and “B”  
05. Captures do not always occur on optional groups  
a) /(A)?B/ matches “AB” and captures “A”  
b) /(A)?B/ matches “B” and does not capture anything  
06. Backreferences is to a group that failed to match  
a) /(A)?B\1/ matches “ABA” but not “B”  
b) Except in JavaScript  
07. Element is optional, group/capture is not optional  
a) /(A?)B/ matches “B” and captures “”  
08. Element is not optional, group/capture is optional  
a) /(A)?B/ matches “B” and does not capture anything  
09.   
  
Regex: /A?B/  
String: "AB B"  
  
  
Captures occur on zero-width matches  
  
Regex: /(A?)B/  
String: "AB B"  
  
Regex: /(A?)B\1/  
String: "ABA B"  
  
Regex: /(A?)B\1C/  
String: "ABAC BC"  
  
  
Captures do not always occur on optional groups  
\*\*\* Except in JavaScript \*\*\*  
  
Regex: /(A)?B/  
String: "AB B"  
  
Regex: /(A)?B\1/  
String: "ABA B"

## **File-03: 85870\_07\_03\_SC11\_findreplace2.mp4**

Description:  
Heading: Find and replace using backreferences  
01. TextMate for Macintosh  
a) <http://macromates.com>  
02. E for Windows  
a) <http://e-texteditor.com>  
03. Create a regular expression that matches target data  
04. Test regular expression and revise as needed  
a) Use anchors and specificity to narrow scope  
05. Add capturing groups  
a) Capture anything that varies row-to-row  
06. Write the replacement string  
a) Use all captures  
b) Add back anything not captured but still needed  
c) May need to use $1 instead of \1  
07.   
  
Write a regex to match target data  
  
Find: /^\d{1,2},[\w. ]+? [\w ]+?,\d{4}/  
  
  
Add capturing groups around anything that varies from row to row  
  
Find: /^(\d{1,2}),([\w. ]+?) ([\w ]+?),(\d{4})/  
  
  
Write the replacement expression.  
Keep all captures and add back anything not captured  
  
Replace: $1,$3,$2,$4

## **File-04: 85870\_07\_04\_SC11\_noncapturing.mp4**

Description:  
Heading: Non-capturing group expressions  
01.

|  |  |
| --- | --- |
| Metacharacter | Meaning |
| ?: | Specify a non-capturing group |

02. Syntax  
a) /(\w+)/ becomes /(?:\w+)/  
03. Turns off capture and backreferences  
a) Optimize for speed  
b) Preserve space for more captures  
04. Support  
a) Most regex engines except Unix tools  
05. /(?:regex)/  
a) ? = “Give this group a different meaning”  
b) : = “The meaning is non-capturing”  
06.   
  
Two capturing groups  
  
Regex: /(oranges) and (apples) to \1/  
String: "oranges and apples to oranges"  
  
  
One non-capturing group and one capturing group  
  
Regex: /(?:oranges) and (apples) to \1/  
String: "oranges and apples to apples"

# **Chapter-8. Lookaround Assertions**

## **File-01: 85870\_08\_01\_SC11\_positivelookahead.mp4**

Description:  
Heading: Positive lookahead assertions  
01.

|  |  |
| --- | --- |
| Metacharacter | Meaning |
| ?= | Positive lookahead assertion |

02. Assertion of what ought to be ahead  
a) If lookahead expression fails, the match fails  
b) Any valid regular expression can be used  
c) Zero-width, does not include group in the match  
03. Support  
a) Most regex engines except Unix  
04. Syntax  
a) /(?=regex)/  
05. Examples  
a) /(?=seashore)sea/ matches “sea” in “seashore” but not “seaside”  
b) Same as /sea(?=shore)/  
06.   
Regex: /(?=seashore)sea/  
String: "seashore seaside"  
  
Regex: /sea(?=shore)/  
String: "seashore seaside"  
  
  
# Contrast lookahead with capturing and non-capturing  
  
Regex: /sea(?:shore)/  
String: "seashore seaside"  
  
Regex: /(sea)shore/  
String: "seashore seaside"  
  
  
# All words followed by a comma  
  
Regex: /\b[A-Za-z']+?\b,/  
String: (see self-reliance.txt)  
  
Regex: /\b[A-Za-z']+?\b(?=,)/  
String: (see self-reliance.txt)  
  
  
# Compare with non-capturing  
  
Regex: /\b[A-Za-z']+?\b(?:,)/  
String: (see self-reliance.txt)  
07.   
Self-Reliance  
By Ralph Waldo Emerson, 1841  
   
"Ne te quÃ¦siveris extra."  
   
"Man is his own star; and the soul that can  
Render an honest and a perfect man,  
Commands all light, all influence, all fate;  
Nothing to him falls early or too late.  
Our acts our angels are, or good or ill,  
Our fatal shadows that walk by us still."  
â€” Epilogue to Beaumont and Fletcher's Honest Man's Fortune.  
  
Cast the bantling on the rocks,  
Suckle him with the she-wolf's teat,  
Wintered with the hawk and fox,  
Power and speed be hands and feet.

## **File-02: 85870\_08\_02\_SC11\_doubletesting.mp4**

Description:  
Heading: Double testing with lookahead assertions  
01. Examples  
a) /(?=seashore)sea/ matches “sea” in “seashore” but not “seaside”  
b) Same as /sea(?=shore)/  
02. Match a pattern that also matches another pattern  
a) /\d{3}-\d{3}-\d{4}/ matches “555-302-4321” and “555-781-6978”  
b) /^[0-5\-]+$/ matches “555-302-4321” and “23140-5”  
c) /(?=^[0-5\-]+$)\d{3}-\d{3}-\d{4}/ matches “555-302-4321”  
d) /(?=^[0-5\-]+$)(?=.\*4321)\d{3}-\d{3}-\d{4}/ matches “555-302-4321”  
03.   
Regex: /\d{3}-\d{3}-\d{4}/  
String: "  
555-302-4321  
555-781-6978  
555-245-1312  
"  
  
# Make sure multiline anchors are enabled.  
  
Regex: /(?=^[0-5\-]+$)\d{3}-\d{3}-\d{4}/m  
String: "  
555-302-4321  
555-781-6978  
555-245-1312  
"  
Regex: /(?=^[0-5\-]+$)(?=.\*4321)\d{3}-\d{3}-\d{4}/m  
String: "  
555-302-4321  
555-781-6978  
555-245-1312  
"  
  
# All words containing a "gh" and followed by a comma.  
  
Regex: \b[A-Za-z']+?\b(?=,)  
String: (see self-reliance.txt)  
  
Regex: \b(?=\w\*gh)[A-Za-z']+?\b(?=,)  
String: (see self-reliance.txt)  
  
  
# Password must be 8-15 characters  
  
Regex: /^.{8,15}$/  
String: "swordfish"  
  
  
# Password must be 8-15 characters and contain 1 digit  
  
Regex: /^(?=.\*\d).{8,15}$/  
String: "sword42fish"  
  
  
# Password must be 8-15 characters and contain 1 digit and 1 capital letter  
  
Regex: /^(?=.\*\d)(?=.\*[A-Z]).{8,15}$/  
String: "sword42Fish"  
04.  
Self-Reliance  
By Ralph Waldo Emerson, 1841  
   
"Ne te quÃ¦siveris extra."  
   
"Man is his own star; and the soul that can  
Render an honest and a perfect man,  
Commands all light, all influence, all fate;  
Nothing to him falls early or too late.  
Our acts our angels are, or good or ill,  
Our fatal shadows that walk by us still."  
â€” Epilogue to Beaumont and Fletcher's Honest Man's Fortune.  
  
Cast the bantling on the rocks,  
Suckle him with the she-wolf's teat,  
Wintered with the hawk and fox,  
Power and speed be hands and feet.

## **File-03: 85870\_08\_03\_SC11\_negativelookahead.mp4**

Description:  
Heading: Negative lookahead assertions  
01.

|  |  |
| --- | --- |
| Metacharacter | Meaning |
| ?! | Negative lookahead assertions |

02. Syntax  
a) /(?!regex)/  
03. Examples  
a) /(?!seashore)sea/ matches “sea” in “seaside” but not “seashore”  
b) Same as /sea(?!shore)/  
c) /online(?! training)/ does not match “online training”  
d) /online(?!.\*training)/ does not match “online video training”  
04.   
Regex: /\bblack\b(?! dog)/  
String: "The black dog followed the black car into the black night."  
  
# Contrasted with the positive lookahead assertion  
Regex: /\bblack\b(?= dog)/  
String: "The black dog followed the black car into the black night."  
  
  
Regex: /(?=^[0-5\-]+$)(?=.\*4321)\d{3}-\d{3}-\d{4}/m  
String: "  
555-302-4321  
555-781-6978  
555-245-1312  
"  
  
Regex: /(?=^[0-5\-]+$)(?!.\*4321)\d{3}-\d{3}-\d{4}/m  
String: "  
555-302-4321  
555-781-6978  
555-245-1312  
"  
  
# All words NOT followed by a comma.  
Regex: /\b[A-Za-z']+?\b(?=,)/  
String: see self-reliance.txt  
  
# All words NOT followed by a comma.  
Regex: /\b[A-Za-z']+?\b(?!,)/  
String: see self-reliance.txt  
  
# All words NOT followed by a comma or period  
Regex: /\b[A-Za-z']+?\b(?![,.])/  
String: see self-reliance.txt  
  
  
# Last occurrence: an item that isn't followed by itself.  
Regex: /\bblack\b(?!.\*\bblack\b)/  
String: "The black dog followed the black car into the black night."  
  
Regex: /(\bblack\b)(?!.\*\1)/  
String: "The black dog followed the black car into the black night."  
05.  
Self-Reliance  
By Ralph Waldo Emerson, 1841  
   
"Ne te quÃ¦siveris extra."  
   
"Man is his own star; and the soul that can  
Render an honest and a perfect man,  
Commands all light, all influence, all fate;  
Nothing to him falls early or too late.  
Our acts our angels are, or good or ill,  
Our fatal shadows that walk by us still."  
â€” Epilogue to Beaumont and Fletcher's Honest Man's Fortune.  
  
Cast the bantling on the rocks,  
Suckle him with the she-wolf's teat,  
Wintered with the hawk and fox,  
Power and speed be hands and feet.

## **File-04: 85870\_08\_04\_SC11\_lookbehind.mp4**

Description:  
Heading: Lookbehind assertions  
01.

|  |  |
| --- | --- |
| Metacharacter | Meaning |
| ?<= | Positive lookbehind assertion |
| ?<! | Negative lookbehind assertion |

02. Assertion of what ought to be behind  
a) Similar to lookahead assertions  
b) If lookbehind expression fails, the match fails  
c) Any valid regular expression can be used  
d) Zero-width, does not include group in the match  
03. Syntax  
a) /(?<=regex)/  
b) /(?<!regex)/  
04. Examples  
a) /(?<=base)ball/ matches the “ball” in “baseball” but not “football”  
b) Same as /ball(?<=baseball)/  
c) /(?<!base)ball/ matches the “ball” in “football” but not “baseball”  
05. Support  
a) Simple expressions in .NET, Java, Perl, PHP, Python, Ruby 1.9  
b) Not supported in JavaScript, Ruby 1.8, Unix  
06. Simple expressions means fixed length  
a) Literal text  
b) Character classes  
c) No repetition or optional expressions  
d) Alternation only with fixed-length items  
 - Allowed: (?<=cat|dog|rat)  
 - Not allowed: (?<=apple|banana|plum)  
07.   
# Will not work in JavaScript!  
  
Regex: /(?<=base)ball/  
String: "I like baseball and football."  
  
Regex: /(?!=base)ball/  
String: "I like baseball and football."  
  
Regex: /(jamin|ny)/  
String: "Benny Benjamin Jenny Lenny"  
  
Regex: /(?<=Ben)(jamin|ny)/  
String: "Benny Benjamin Jenny Lenny"  
  
Regex: /(?<=Ben|Jen)(jamin|ny)/  
String: "Benny Benjamin Jenny Lenny"  
  
Regex: /(?<!Ben|Jen)(jamin|ny)/  
String: "Benny Benjamin Jenny Lenny"

## **File-05: 85870\_08\_05\_SC11\_powerpositions.mp4**

Description:  
Heading: The power of positions  
01. Allows testing of a regular expression apart from matching  
a) Peek forwards or backwards  
 - /sea(?=shore)/  
b) Match a string using multiple expressions  
 - /^(?=.\*\d)(?=.\*[A-Z]).{8,15}$/  
c) Define rejection expressions  
 - /online(?! training)/  
d) Find last occurrence  
 - /(item)(?!.\*\1)/  
02. Zero-width means zero position movement  
a) /(?=seashore)sea/ matches “sea” in “seashore”  
b) /(?<![$\d])\d+\.\d\d/ matches “54.00” but not “$54.00”  
c) /(?<![$\d])(?=\d+\.\d\d)/  
 - What gets matched?  
 - Where is the regular expression engine pointer?  
d) Useful for inserting text (using find and replace)  
03.   
  
# These will not work in JavaScript!  
  
Regex: /(?<![$\d])\d+\.\d\d/  
String: "This costs 53.00 or $54.00."  
  
  
# Insertion by using all assertions  
  
Regex: /(?<![$\d])(?=\d+\.\d\d)/  
String: "This costs 53.00 or $54.00."  
Replace: "$"  
  
  
# Adding commas to delimit a number  
  
String: "An astronomical unit is 149597870.7 kilometers, approximately the average distance between the Sun and Earth."  
  
Regex: /(\d\d\d)+/  
Regex: /(?<=\d)(\d\d\d)+(?!\d)/  
Regex: /(?<=\d)(?=(\d\d\d)+(?!\d))/  
Replace: ","

# **Chapter-9. Unicode and Multibyte Characters**

## **File-01: 85870\_02\_06a\_SC11\_unicode2.mp4**

Description:  
Heading: About Unicode  
01. Single byte  
a) Uses one byte (eight bits) to represent a character  
b) Allows for 256 characters  
c) A-Z, a-z, 0-9, punctuation, common symbols  
02. Double byte  
a) Uses two bytes (16 bits) to represent a character  
b) Allows for 65,536 characters  
03. Many more characters than English alphabet  
a) Latin: à á â ã ä å  
b) Symbols: ≤ ≥ ≠ € £  
c) Arabic, Chinese, Greek, Hebrew, Korean, Thai  
d) Over 109,000 characters  
04. Unicode  
a) Variable byte size  
b) Maintains compatibility with one- and two-byte encoding  
c) Allows for over one million characters  
05. Unicode  
a) Mapping between a character and a number  
b) “U+” followed by a four-digit hexadecimal number  
 - ∞ is written as U+221E  
c) Combinations  
 - é can be U+00E9 or U+0065 U+0301  
 - Can combine more than two

## **File-02: 85870\_02\_06b\_SC11\_unicode2.mp4**

Description:  
Heading: Unicode in regular expressions  
01. Complications for regular expressions  
a) Word can be spelled multiple ways  
 - “cafe”, “café”  
b) Words can be encoded multiple ways  
 - “café” can be encoded as four or five characters  
c) Wildcard matching  
d) Backtracking  
e) Unicode is relatively new  
02. Unicode indicator: \u  
a) \u followed by a four-digit hexadecimal number (0000-FFFF)  
b) /caf\u00E9/ matches “café” but not “cafe”  
03. Support  
a) Java, JavaScript, .NET, Python, Ruby  
b) Perl and PHP use \x instead  
c) Not supported in older Unix tools  
04.   
(Mac users can access special characters from "Edit > Special Characters")  
  
Regex: /cafe/  
String: "cafe cafÃ© cafÃ©"  
  
Regex: /cafÃ©/  
String: "cafe cafÃ© cafÃ©"  
  
Regex: /caf\u00E9/  
String: "cafe cafÃ© cafÃ©"  
  
Regex: /caf\u0065\u0301/  
String: "cafe cafÃ© cafÃ©"

## **File-03: 85870\_02\_07\_SC11\_unicodewildcard.mp4**

Description:  
Heading: Unicode wildcard and properties  
01. Unicode wildcard: \X  
a) Matches any single character  
b) Always matches line breaks (like /./s)  
c) /caf\X/ matches “café” and “cafe”  
02. Support  
a) Only supported in Perl and PHP  
03. Unicode property: \p{property}  
a) Matches characters that have a property  
b) /\p{Mark}/ or /\p{M}/ matches any “mark” (accents)  
c) /\p{Letter}/ or /\p{L}/ matches any letter  
04.

|  |  |
| --- | --- |
| Unicode property | Abbreviation |
| Letter | L |
| Mark | M |
| Separator | Z |
| Symbol | S |
| Number | N |
| Punctuation | P |
| Other | C |

05. Unicode not-property: \P{property}  
a) Matches characters that do not have a property  
b) /caf\P{M}\p{M}/ matches “café”  
06. Support  
a) Java, .NET, Perl, PHP, Ruby  
07.   
  
\X is only supported in Perl and PHP  
Not supported in JavaScript  
  
Regex: /caf\X/  
String: "cafe cafÃ© cafÃ©"  
  
  
\p and \P are only supported in Java, .NET, Perl, PHP, Ruby  
Not supported in JavaScript  
  
Regex: /caf\P{M}\p{M}/  
String: "cafe cafÃ© cafÃ©"

# **Chapter-10. Useful Regular Expressionsss**

## **File-01: 85870\_09\_01\_SC11\_howtouse.mp4**

Description:  
Heading: How to use this chapter  
01. Not one-size-fits-all regular expressions  
02. Any regular expression can be written broadly or narrowly  
a) Broad: permissive  
b) Narrow: restrictive, brittle  
03. Regular expression to match a year  
a) /\d{4}/ matches 2005, but also 0000-9999  
b) /(19|20)\d\d/ matches 1900-2099  
c) /(19[5-9]\d|20[0-4]\d)/ matches 1950-2049  
04. Never use someone else’s regular expression without checking it carefully and fine-tuning it for your specific purpose.  
05. How to write or customize a regular expression  
a) Examine the data to be matched  
b) Determine what aspects of the data are important  
c) Determine what level of precision is required  
d) Make a list of “edge cases” to test  
 - Longest, shortest  
 - Highest, lowest  
 - Most unusual, most oddly-formatted  
06. Use anchors, delimiters, or context  
a) /\w+/ matches “%^@X&\*!”  
b) /^\w+$/  
c) /\b\w+\b/  
d) / \w+ /  
e) /,\w+,/  
f) /and \w+\./  
07. Be mindful of greediness and laziness.

## **File-02: 85870\_09\_02\_SC11\_matchnames.mp4**

Description:  
Heading: Match names  
01.   
Regex: /\w+/  
String: "  
Kevin  
Lynda  
Charlie  
Lucy  
Linus  
Sally  
"  
  
Regex: /^\w+$/m  
String: "$kevin"  
  
Regex: /^[A-Za-z]+$/m  
String: "0kevin"  
  
# Does first letter have to be capitalized?  
Regex: /^[A-Z][A-Za-z]+$/m  
String: "kevin"  
  
Regex: /^[A-Z][A-Za-z.'\-]+$/m  
String: "J.R."  
  
# Match first and last name  
Regex: /^[A-Z][A-Za-z.'\- ]+$/m  
String: "George Washington"  
  
# Capture first and last name  
Regex: /^([A-Z][A-Za-z.'\-]+) ([A-Z][A-Za-z.'\-]+)$/m  
String: "George Washington"  
  
# Capture first, middle, and last name  
Regex: /^([A-Z][A-Za-z.'\-]+) (?:([A-Z][A-Za-z.'\-]+) )?([A-Z][A-Za-z.'\-]+)$/m  
String: "John Quincy Adams"  
  
# Tough call  
String: "Martin Van Buren"

## **File-03: 85870\_09\_03\_SC11\_matchpostal.mp4**

## **File-04: 85870\_09\_04\_SC11\_matchemails.mp4**

## **File-05: 85870\_09\_05\_SC11\_matchurls2.mp4**

## **File-06: 85870\_09\_06\_SC11\_decimal2.mp4**

## **File-07: 85870\_09\_07\_SC11\_ipaddress.mp4**

## **File-08: 85870\_09\_08\_SC11\_matchdates.mp4**

## **File-09: 85870\_09\_09\_SC11\_matchtimes.mp4**

## **File-10: 85870\_09\_10\_SC11\_matchtags.mp4**

## **File-11: 85870\_09\_11\_SC11\_matchpasswords3.mp4**

## **File-12: 85870\_09\_12\_SC11\_matchcreditcards.mp4**

## **File-13: 85870\_09\_13\_SC11\_wordsnearwords.mp4**

## **File-14: 85870\_09\_14\_SC11\_formatfirst.mp4**

## **File-15: 85870\_09\_15\_SC11\_formatsecond.mp4**

## **File-16: 85870\_09\_16\_SC11\_formatthird.mp4**

# **Chapter-11. Conclusion**

## **File-01: 85870\_10\_01\_SC11\_conclusion.mp4**