CompSci 220

Programming Methodology

19: Introduction to Regular Expressions

based on slides by Mehryar Mohri, NYU

Objectives

- Learn Regular Expressions
  - What are they?
  - Pattern matching text.
  - Meta characters.
  - Allow you to search for text in files
- The **grep** command

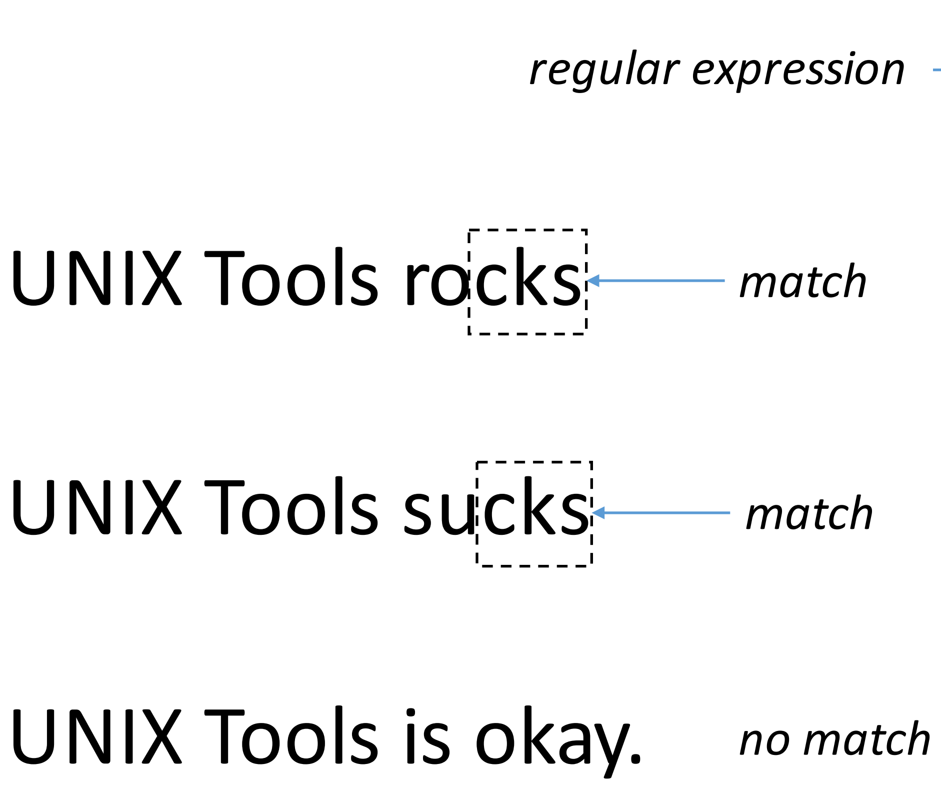
What is a Regular Expression?

- A regular expression (*regex*) describes a set of possible input strings.
- *Regular expressions* descend from a fundamental concept in Computer Science called *finite automata* theory.
- *Regular expressions* are endemic to Unix
  - **Vim**, **ed**, **sed**, and **emacs**
  - **awk**, **tcl**, **perl**, and **python**
  - **grep**, **egrep**, **fgrep**
  - **compilers**

Regular Expressions

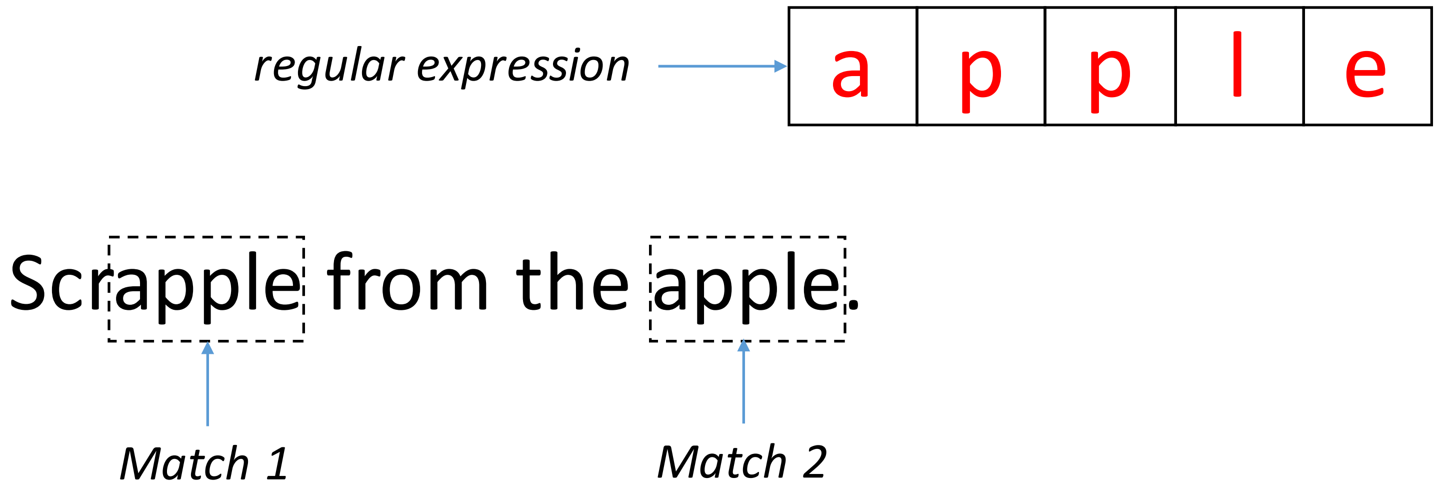
- The simplest regular expressions are a string of literal characters to match.
- The string ***matches*** the regular expression if it contains the substring.

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP


regular expression
c
k
s
UNIX Tools rocks.
match
UNIX Tools sucks.
match
UNIX Tools is okay.
no match

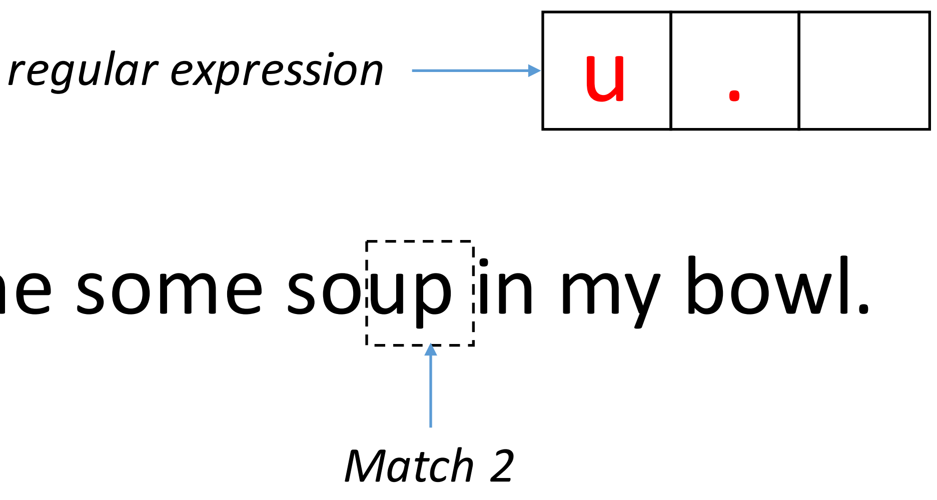
Regular Expressions

- A regular expression can match a string in more than one place.



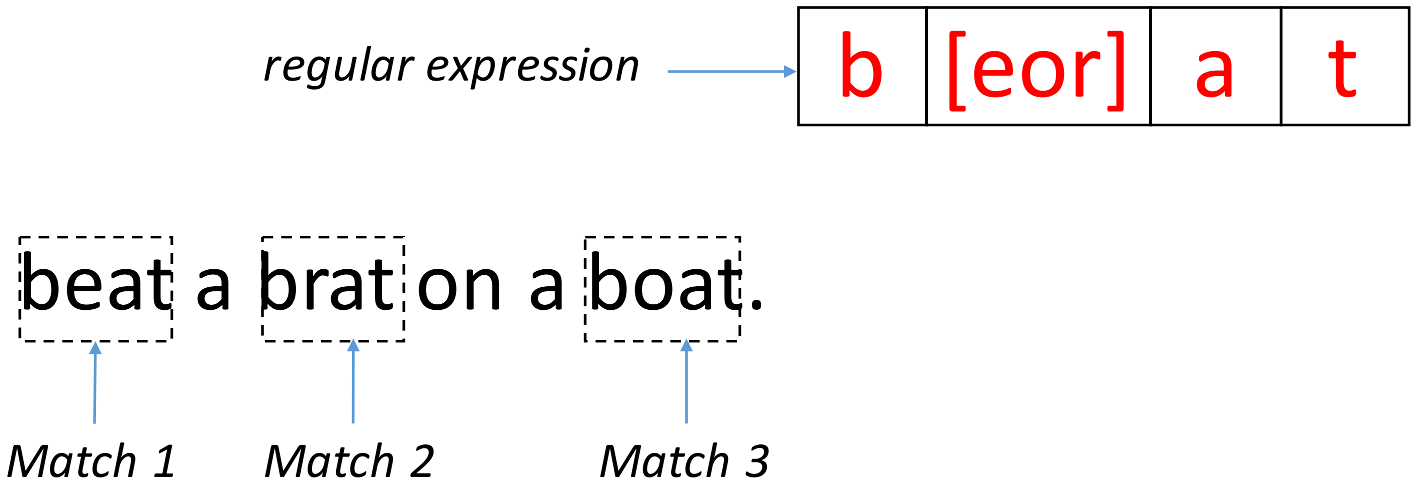
Regular Expressions Meta Characters

- The **.** regular expression can be used to match **any** character.



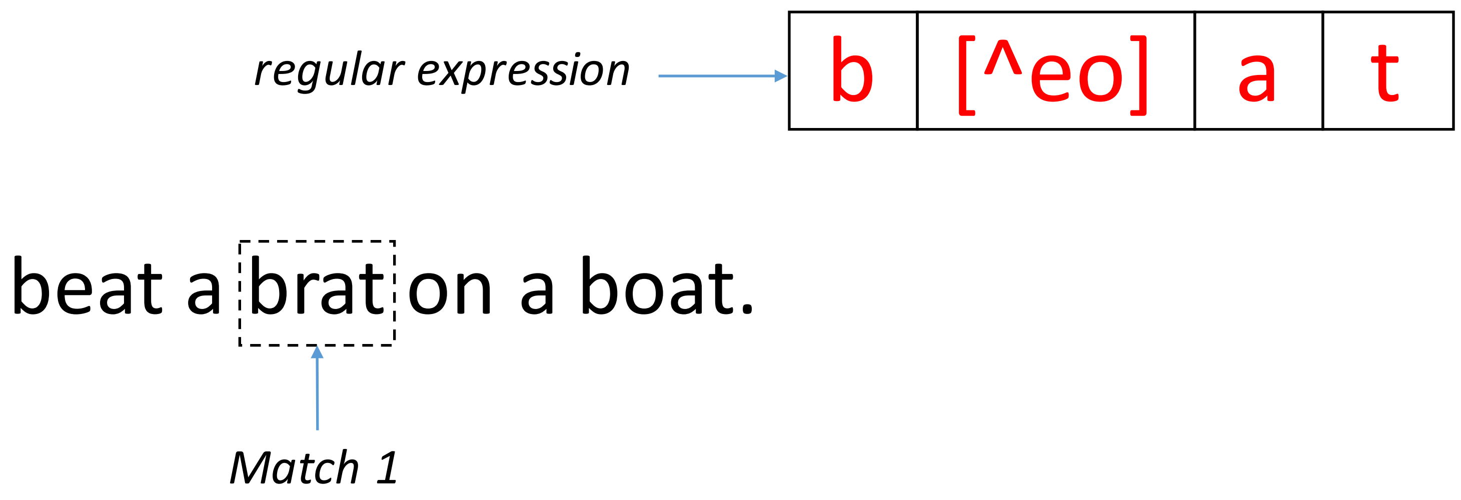
Regular Expressions Character Classes

- Character classes **[ ]** can be used to match any specific set of characters.



Regular Expressions Character Classes

- Character classes can be negated with the **[^]** syntax.



More About Character Classes

- Basics
    - [aeiou] will match any of the characters a, e, i, o, or u
    - [mN]ohri will match mohri or Mohri
- Ranges can also be specified in character classes
    - [1-9] is the same as [123456789]
    - [abcde] is equivalent to [a-e]
- You can combine them
    - [abcde123456789] is equivalent to [a-e1-9]
- Note that the – character has a special meaning in a character class **but only** if it is used in a range.
    - [-123] would match the characters -, 1, 2, or 3.

Named Character Classes

- Commonly used character classes can be referred to by name:
  - alpha, lower, upper, alnum, digit, punct, cntrl

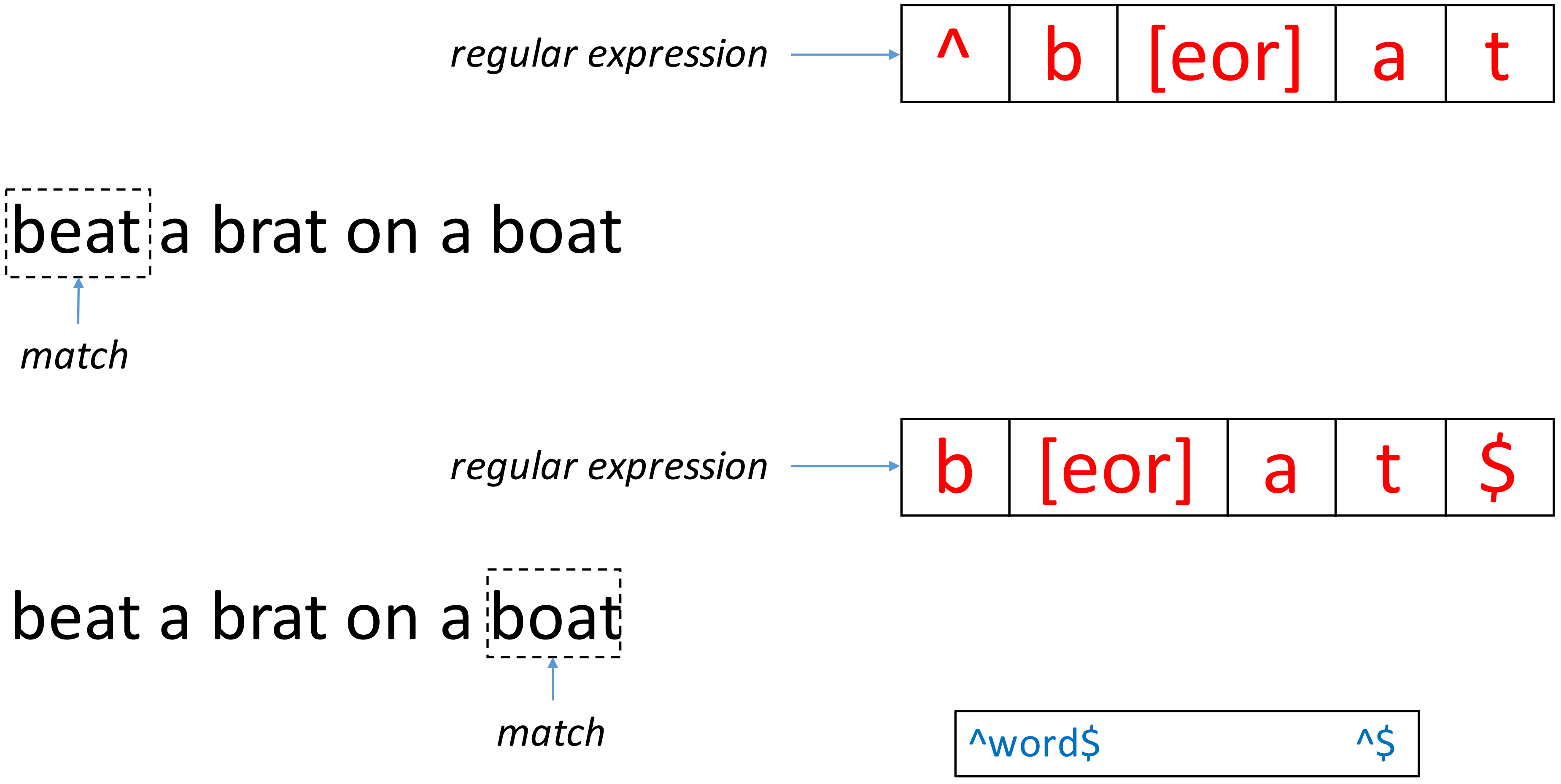
- Syntax: [:name:]

[a-zA-Z]	[[:alpha:]]
[a-zA-Z0-9]	[[:alnum:]]
[45a-z]	[45[:lower:]]

- Important for portability across languages.

Anchors

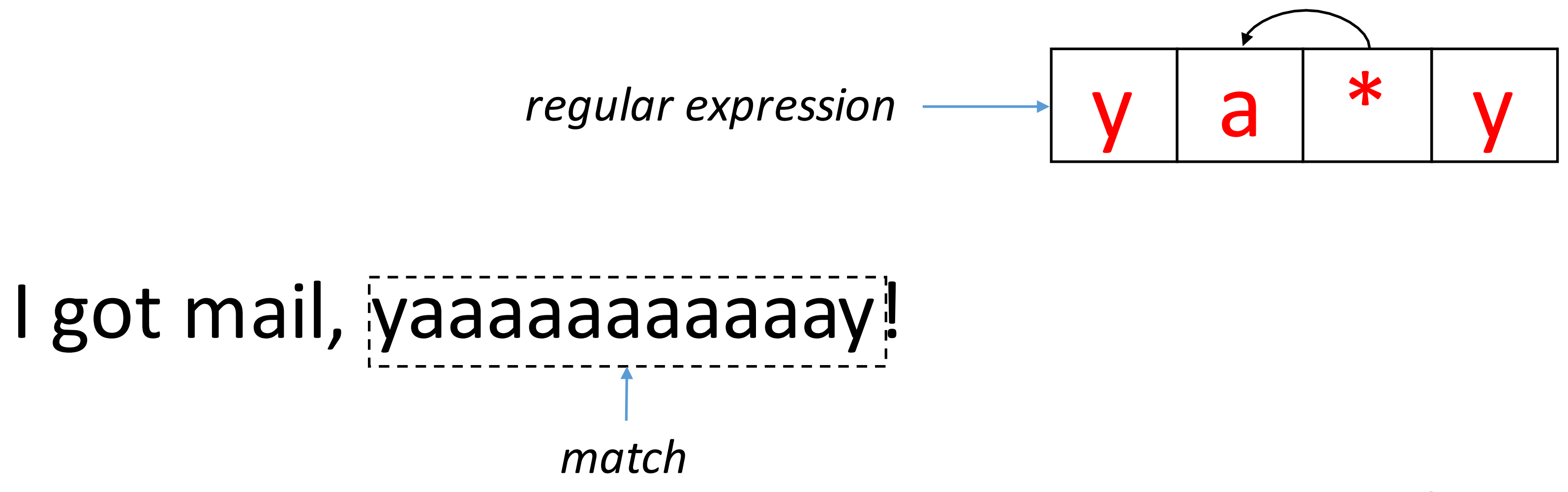
- Anchors are used to match at the beginning or end of a line (or both).
- ^ means beginning of the line.
- $ means end of the line.

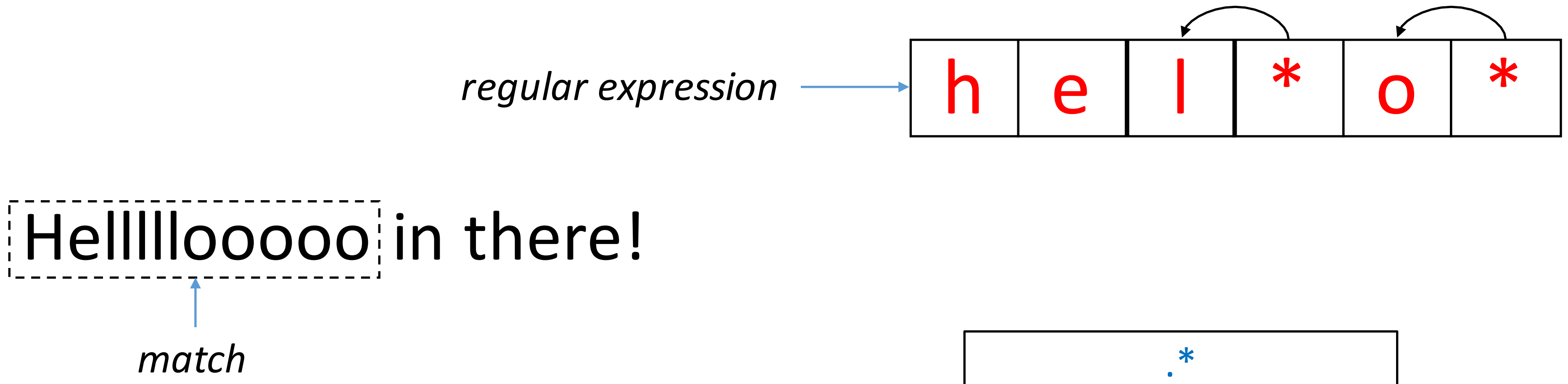


useful RE patterns

Repetition

- The * is used to define **zero or more** occurrences of the *single* regular expression preceding it.





.*

useful RE patterns

Repetition Ranges

- Ranges can also be specified
  - { } notation can specify a range of repetitions for the immediately preceding regular expression.
  - {n} means exactly n occurrences.
  - {n, } means at least n occurrences.
  - {n,m} means at least n occurrences but not more than m occurrences.

- Example:
  - .{0,} same as .*
  - A{2,} same as aaa*

Subexpressions

- If you want to group part of an expression so that * or { } applies to more than just the previous character, use ( ) notation.

- Subexpressions are treated like a single character.
  - a* matches 0 or more occurrences of a.
  - abc* matches ab, abc, abcc, abccc, …
  - (abc)* matches abc, abcabc, abcabcabc, …
  - (abc){2,3} matches abcabc or abcabcabc

grep

- **grep** comes from the ed (Unix text editor) search command "**g**lobal **r**egular **e**xpression **p**rint" or **g/re/p**.
- This was such a useful command that it was written as a standalone utility.
- There are two other variants, **egrep** and **fgrep**, that comprise the *grep* family.
- *grep* is the answer to the moments where you know you want the file that contains a specific phrase but you can't remember its name.

Family Differences

- **grep** – uses regular expressions for pattern matching.
- **fgrep** – file grep, does not use regular expressions, only matches fixed strings but can get search strings from a file.
- **egrep** – extended grep, uses a more powerful set of regular expressions (but does not support back referencing), generally the fastest member of the grep family.
- **agrep** – approximate grep, not standard.

Syntax

- Regular expression concepts we have seen so far are common to **grep** and **egrep**.

- grep and egrep have different syntax

- Major syntax differences:
  - **grep**: \(and \), \{ and \}
  - **egrep**: (and), { and }

Protecting Regex Meta Characters

- Since many of the special characters used in regexs also have special meaning to the shell, it’s a good idea to get in the habit of single quoting your regexs.
  - This will protect any special characters from being operated on by the shell.
  - If you habitually do it, you won’t have to worry about when it is necessary.

Escaping Special Characters

- Even though we are single quoting our regexs so the shell won’t interpret the special characters, some characters are special to **grep** (e.g., * and .)
- To get literal characters, we *escape* the character with \ (backslash)

- Suppose we want to search for the character sequence ‘a*b*’
  - Unless we do something special, this will match zero or more ‘a’s followed by zero or more ‘b’s, *not what we want*.

  - ‘a*b*’ will fix this – now the asterisk is treated as regular characters.

egrep: Alternation

- Regex also provides an alternation character | for matching one or another expression.
  - (T|Fl)an will match 'Tan' or 'Flan'
  - ^(From|Subject): will match the From and Subject lines of a typical email *It matches a beginning line followed by either the characters 'From' or 'Subject' followed by a ':'*

- Subexpressions are used to limit the scope of alternation
  - At(ten|nine)tion then matches "Attention" or "Atninetion", not "Attion" or "ninetion" as would happen without the parenthesis: Atten|ninetion

egrep: Repetition Shorthands

- The * (star) has already been seen to specify zero or more occurrences of the immediately preceding character

- The + (plus) means “one more more”
  - abc+d will match abcd, abccd, abccccccccd, etc.
  - But, it will not match abcd
  - Equivalent to {1,}

egrep: Repetition Shorthands

- The ? (question mark) specifies an optional character, the single character that immediately precedes it.
  - July? will match Jul or July
  - Equivalent to {0,1}
  - Also equivalent to (Jul|July)
- The *, ?, and + are known as *quantifiers* because they specify the quantity of a match.
- Quantifiers can also be used with subexpressions:
  - (a*c)+ will match c, ac, aac, or aacaacac
  - But, will not match a or the blank line

egrep: Back References

- Sometimes it is handy to be able to refer to a match that was made earlier in a regular expression.
- This is does using *backreferences*:
  - `\n` is the back reference specifier, where n is a number
- Looks for the nth subexpression
- For example, to find if the first word of a line is the same as the last:
  - `^([[:alpha:]]{1,}) .* \1$`
  - The `^([[:alpha:]]{1,})` matches one or more letters

Practical Regular Expressions

- Variable names in C- or Java-like languages
  - `[a-zA-Z_][a-zA-Z_0-9]*`

- Dollar amount with optional cents
  - `\$[0-9]+(\.[0-9][0-9])?`

- Time of day
  - `(1[012]|[1-9]):[0-5][0-9] (am|pm)`

- HTML headers <h1>, <H1>, <h2>, …
  - `<[hH][1-6]>`

Exercise

- Write a regular expression matching all words with an upper case Z.

Exercise

- Write a regular expression matching all words with an upper case Z.

```
egrep ‘Z’ words.txt
```

Exercise

- Write a regular expression matching all words that begin with an upper case Z at the start of a line.

Exercise

- Write a regular expression matching all words that begin with an upper case Z at the start of a line.

```
egrep ‘^Z’ words.txt
```

Exercise

- Write a regular expression matching all words that begin with an upper case Z at the end of a line.

Exercise

- Write a regular expression matching all words that begin with an upper case Z at the end of a line.

```
egrep ‘Z$’ words.txt
```

Exercise

- Write a regular expression that begins with a ‘t’ followed by a single character followed by an ‘m’ at the end of the line

Exercise

- Write a regular expression that begins with a ‘t’ followed by a single character followed by an ‘m’ at the end of the line

```
egrep ‘t.m$’ words.txt
```

Exercise

- Write a regular expression that matches all words of exactly length 4.

Exercise

- Write a regular expression that matches all words of exactly length 4.

```
egrep '^....$'words.txt

egrep '^.{4}$'words.txt
```

Exercise

- Write a regular expression that matches all words that begin with a ‘t’ and end with an ‘m’ and are exactly 5 characters long.

Exercise

- Write a regular expression that matches all words that begin with a ‘t’ and end with an ‘m’ and are exactly 5 characters long.

```
egrep ‘^t.{3}m$’ words.txt
```

Exercise

- Write a regular expression that matches all words that begin with a ‘q’, followed by 0 or more characters, followed by one or more ‘z’, followed by zero or more characters, followed by one or more ‘l’ (lowercase L), followed by a ‘y’ as the last character

Exercise

- Write a regular expression that matches all words that begin with a ‘q’, followed by 0 or more characters, followed by one or more ‘z’, followed by zero or more characters, followed by one or more ‘l’ (lowercase L), followed by a ‘y’ as the last character

```
egrep ‘q.*z+.*l+y$’ words.txt
```

Exercise

- Write a regular expression that matches all words that start with either an ‘a’, ’b’, or ‘c’, followed by at least one vowel, followed by ‘x’, ‘y’, or ‘z’ at the end of the line.

Exercise

- Write a regular expression that matches all words that start with either an ‘a’, ’b’, or ‘c’, followed by at least one vowel, followed by ‘x’, ‘y’, or ‘z’ at the end of the line.

```
egrep ‘^[a-c][aeiou]+[x-z]$’ words.txt
```

Exercise

- Write a regular expression that matches all words that begin with the words ‘dog’ or ‘cat’, followed by any character, followed by ‘th’ or ‘ng’ at the end of the line.

Exercise

- Write a regular expression that matches all words that begin with the words ‘dog’ or ‘cat’, followed by any character, followed by ‘th’ or ‘ng’ at the end of the line.

```
egrep ‘^(dog|cat).*(th|ng)$’ words.txt
```