Unix Commands

Paste: merge lines of files

```
$ paste file1 file2
```

Use any delimiter such as ‘-’ in between:

```
$ paste –d - file1 file2
```

gzip: compressing the files

gzip – Reduce the size of a file.

```
$ ls –l new.txt
-rw-r--r-- 1 shivram_2 None 85 Aug 27 15:37 new.txt

$ gzip new.txt

$ ls –l new.txt.gz
-rw-r--r-- 1 shivram_2 None 68 Aug 27 15:37 new.txt.gz

$ gunzip new.txt.gz #---- To expand the file
```

sort

- Syntax: *sort [-fnr+x] [-o filename] [filename(s)]*

-f	Ignore case (fold into lower case)
-n	Numeric order
-r	Sort in reverse order
+x	Ignore first x fields when sorting

-o filename - write output to filename, filename can be the same as one of the input files

uniq: list UNIQue items

- Remove or report adjacent duplicate lines
- Syntax: *uniq [-cdu] [input-file] [output-file]*
  - **`-c`** Supersede the -u and -d options and generate an output report with each line preceded by an occurrence count
  - **`-d`** Write only the duplicated lines
  - **`-u`** Write only those lines which are not duplicated
  - The default output is the union (combination) of -*d* and -*u*

tr: TRanslate Characters

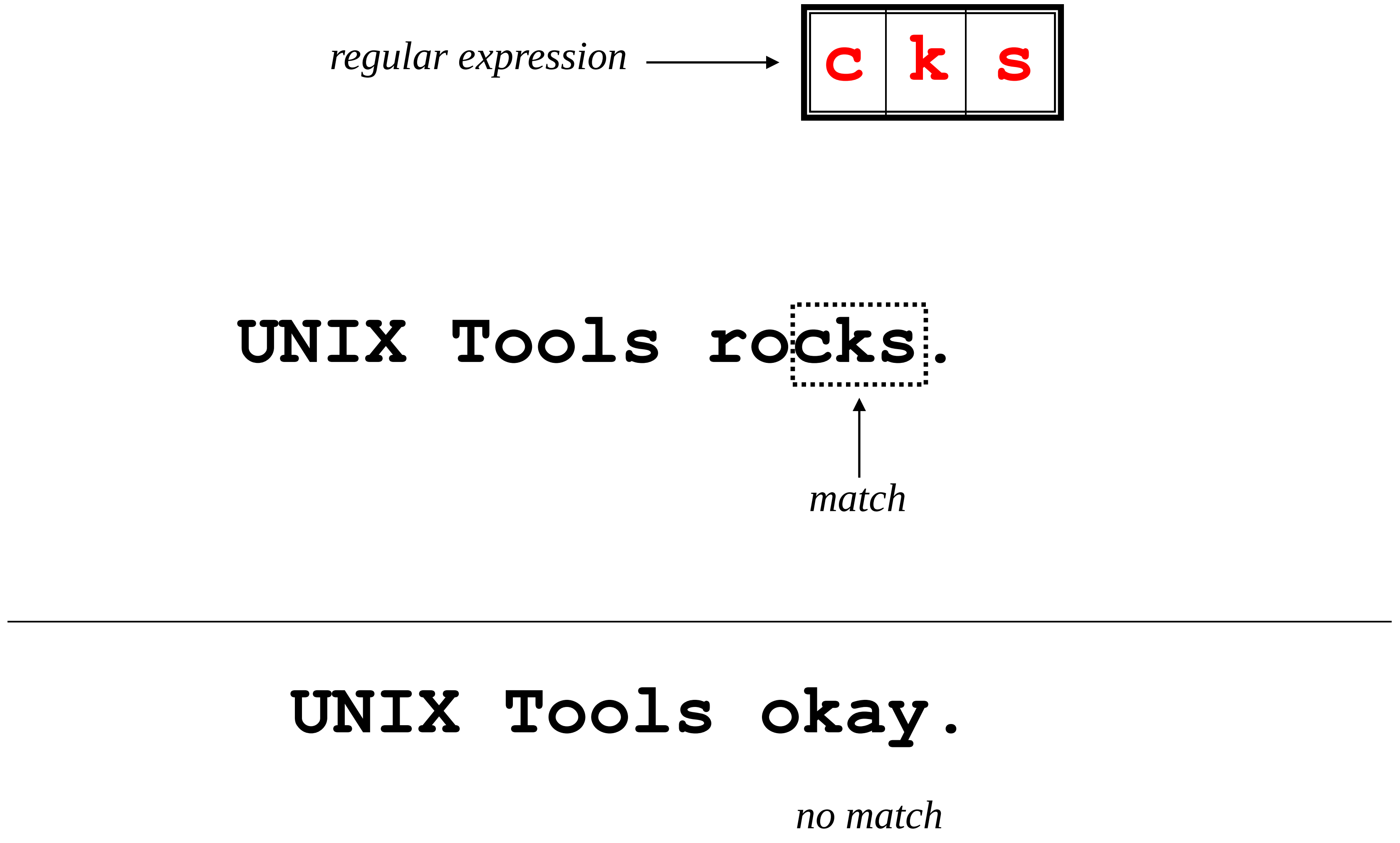
- *tr* reads from standard input.
  - Any character that does not match a character in *string1* is passed to *standard output* unchanged
  - Any character that does match a character in *string1* is translated into the corresponding character in *string2* and then passed to *standard output*
- Examples
  - *tr s z* replaces all instances of *s* with z
  - *tr so zx* replaces all instances of *s* with *z* and *o* with *x*
  - *tr a-z A-Z* replaces all lower case characters with upper case characters
  - *tr –d a-c* deletes all a-c characters

Regular Expression

- A regular expression (*regex*) describes a set of possible input strings.

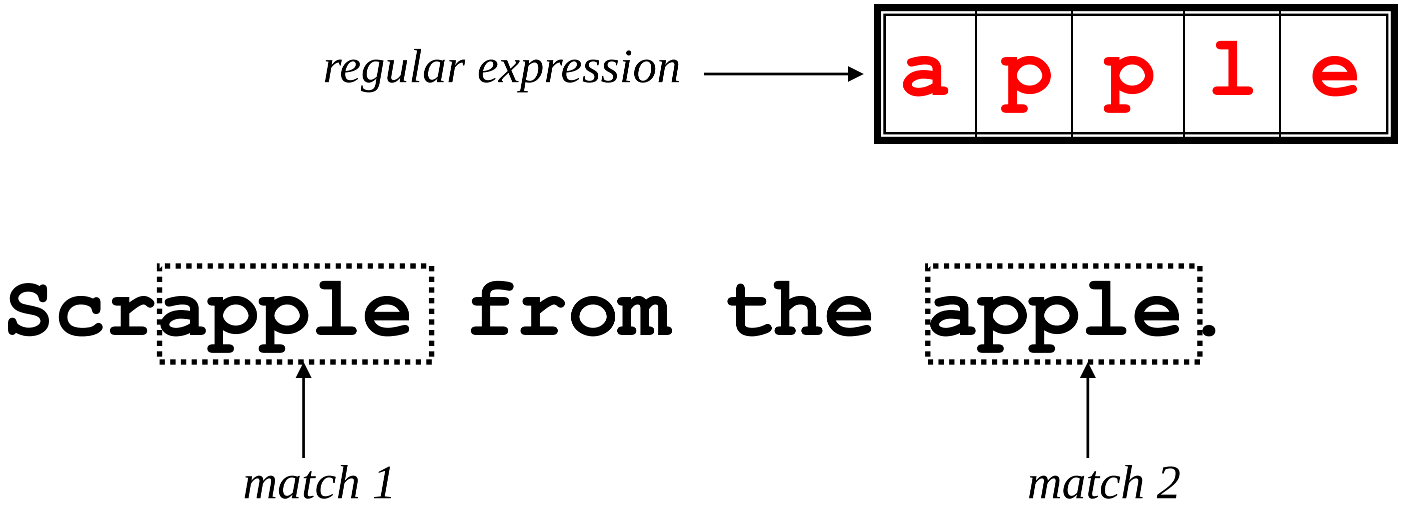
- The string ***matches*** the regular expression if it contains the substring.

- *Regular expressions* are endemic to Unix
  - **vi**, **ed**, **sed**, and **emacs**
  - **awk**, **tcl**, **perl** and **Python**
  - **grep**, **egrep**, **fgrep**
  - **compilers**


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

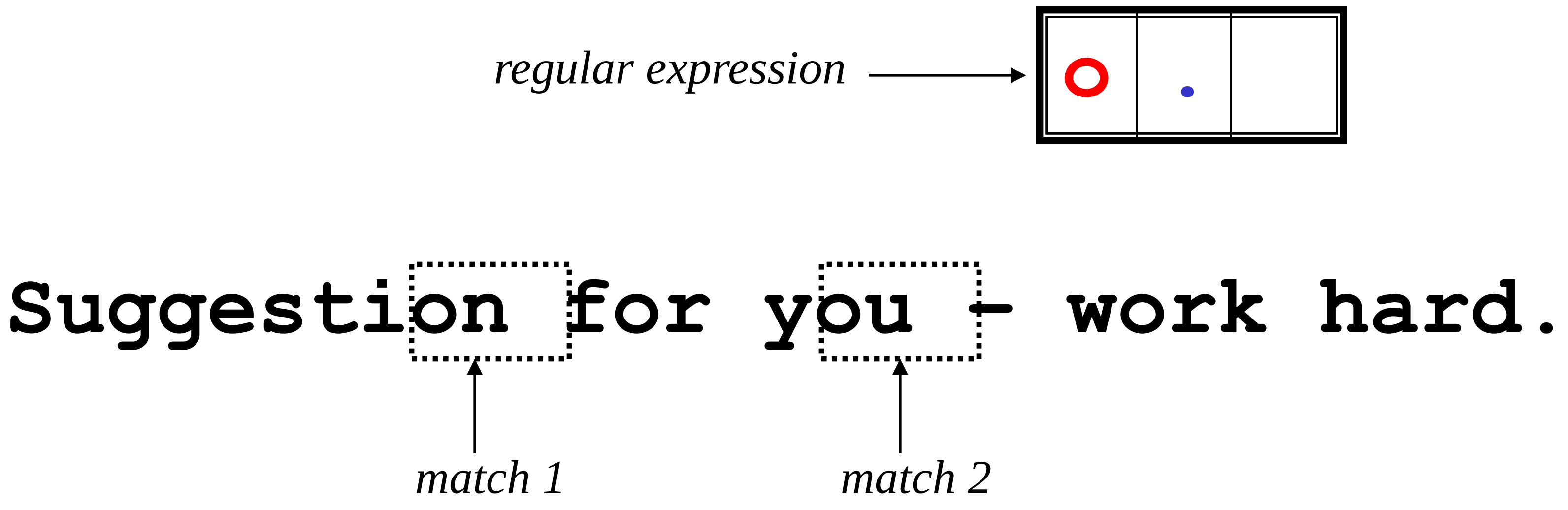
Regular Expressions

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



Regular Expressions

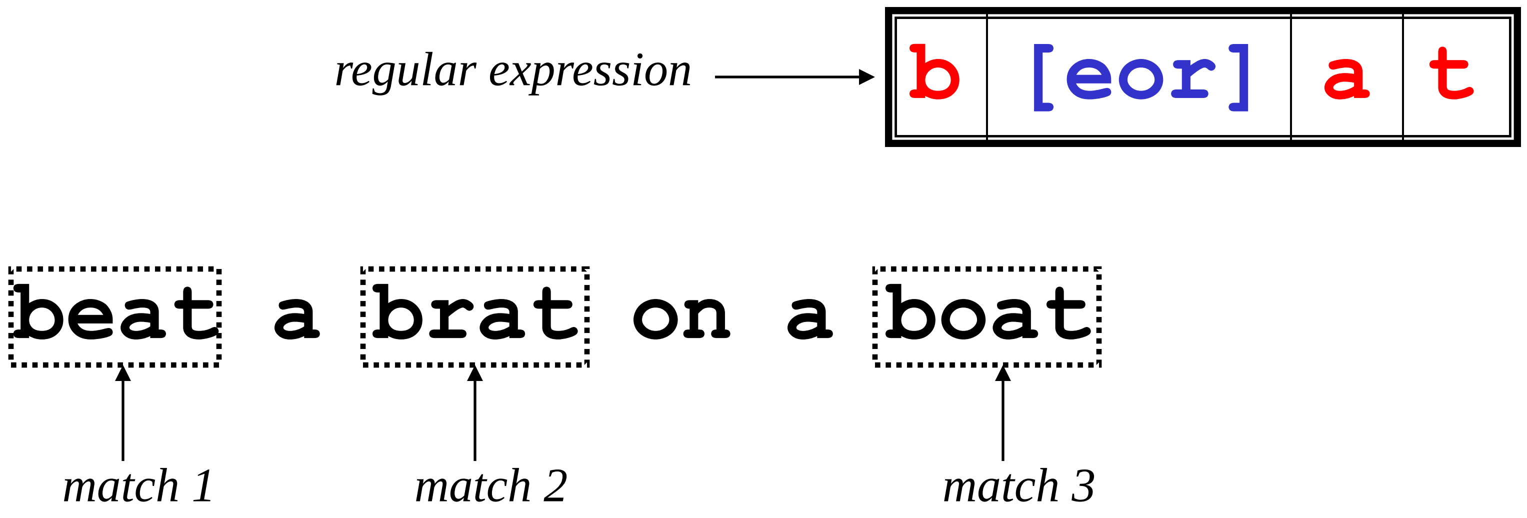
- The **.** regular expression can be used to match any character.



```
Suggestion for you - work hard.
```

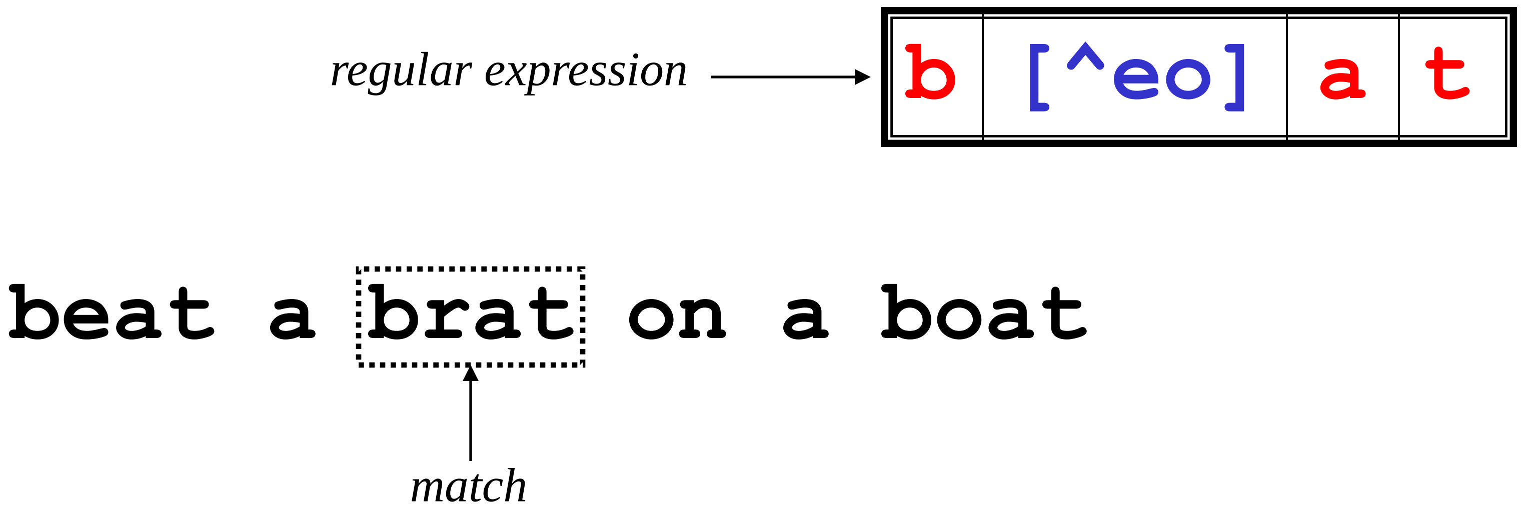
Character Classes

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



Negated Character Classes

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



More About Character Classes

  - **`[aeiou]`** will match any of the characters **`a`**, **`e`**, **`i`**, **`o`**, or **`u`**
  - **`[kK]orn`** will match **`korn`** or **`Korn`**
- Ranges can also be specified in character classes
  - **`[1-9]`** is the same as **`[123456789]`**
  - **`[a-e]`** is equivalent to **`[abcde]`**
  - Multiple ranges can be combined also
    - **`[a-e1-9]`** is equivalent to **`[abcde123456789]`**
  - Note that the **`-`** character has a special meaning in a character class ***but only*** if it is used within a range, **`[-123]`** would match the characters **`-`**, **`1`**, **`2`**, or **`3`**

Named Character Classes

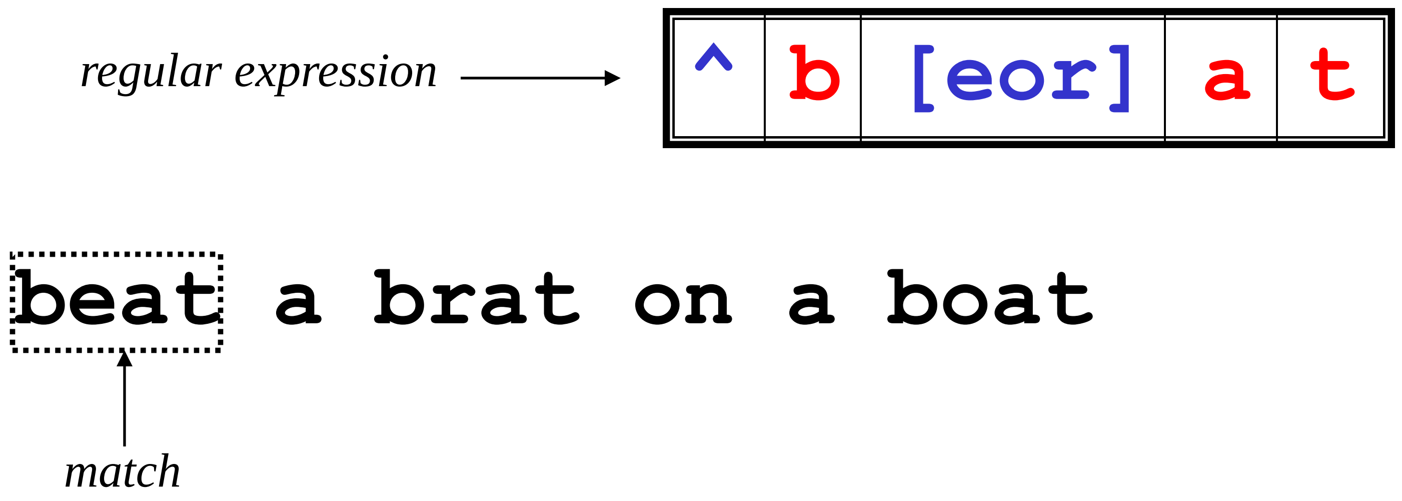
- Commonly used character classes can be referred to by name (*alpha*, *lower*, *upper*, *alnum*, *digit*, *punct*, *cntrl*)
- Syntax `[:`***name***`:]`

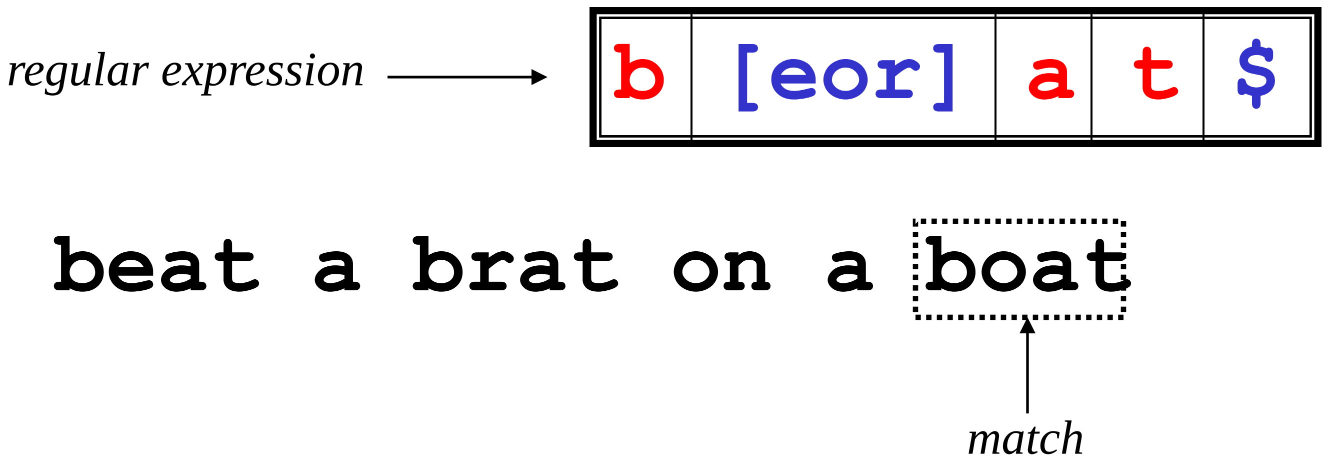
– `[a-zA-Z]`	`[[:alpha:]]`
– `[a-zA-Z0-9]`	`[[:alnum:]]`
– `[45a-z]`	`[45[:lower:]]`

- Important for portability across languages

Anchors

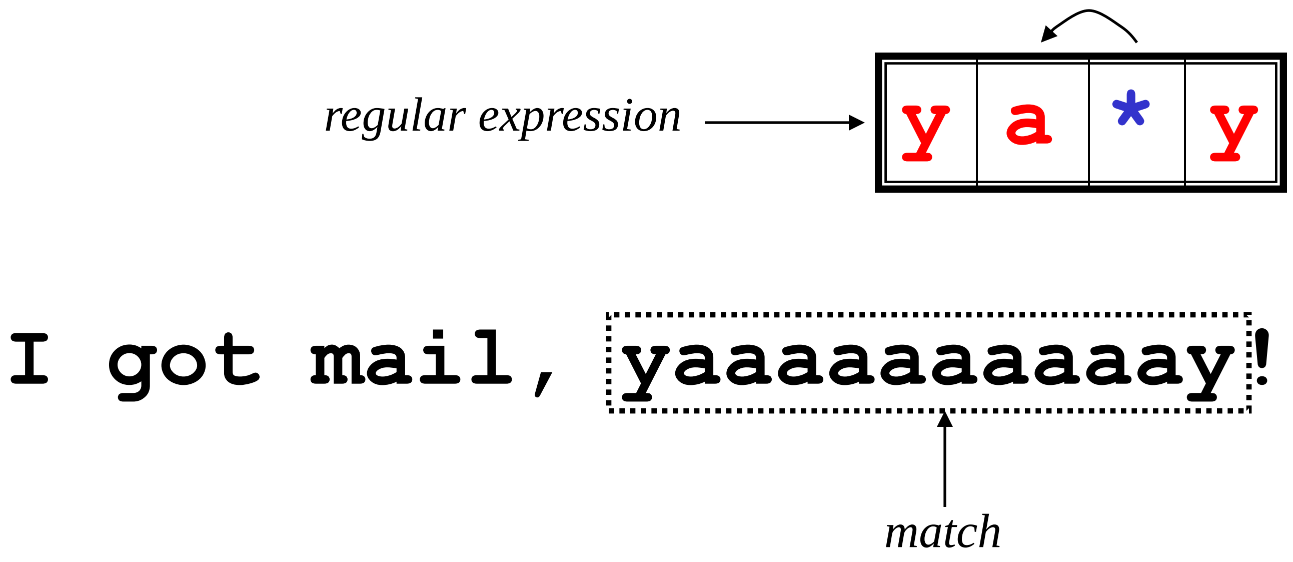
- Anchors are used to match at the beginning or end of a line (or both).
- **^** means beginning of the line
- **$** means end of the line

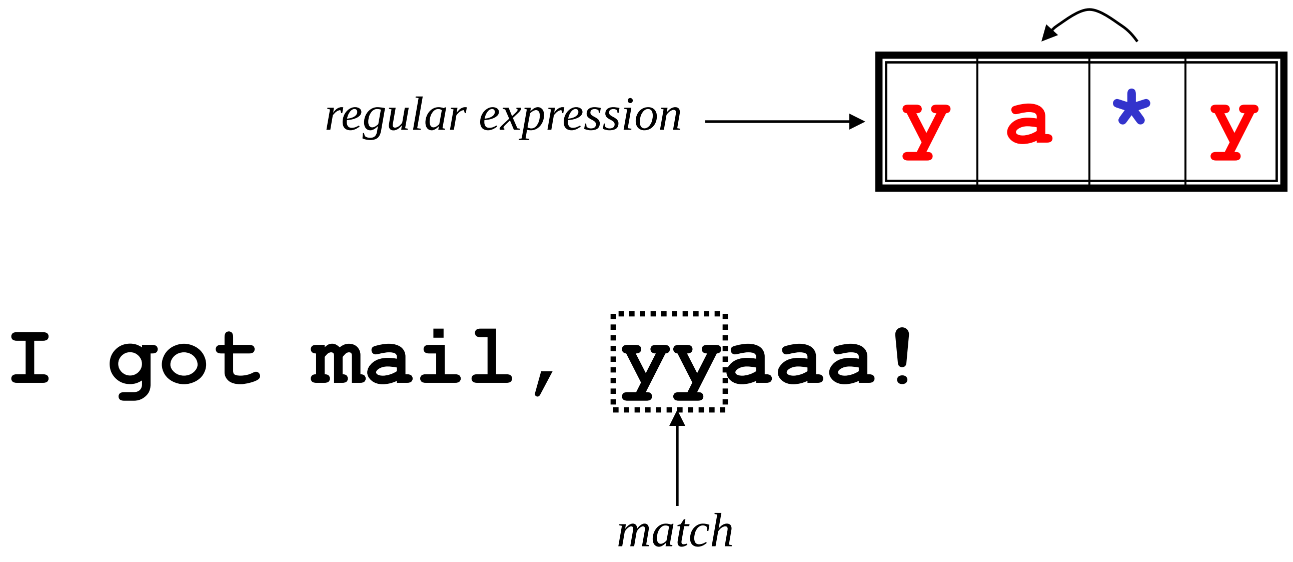

regular expression
^ b [eor] a t
beat a brat on a boat
match


regular expression
b [eor] a t $
beat a brat on a boat
match

Repetition

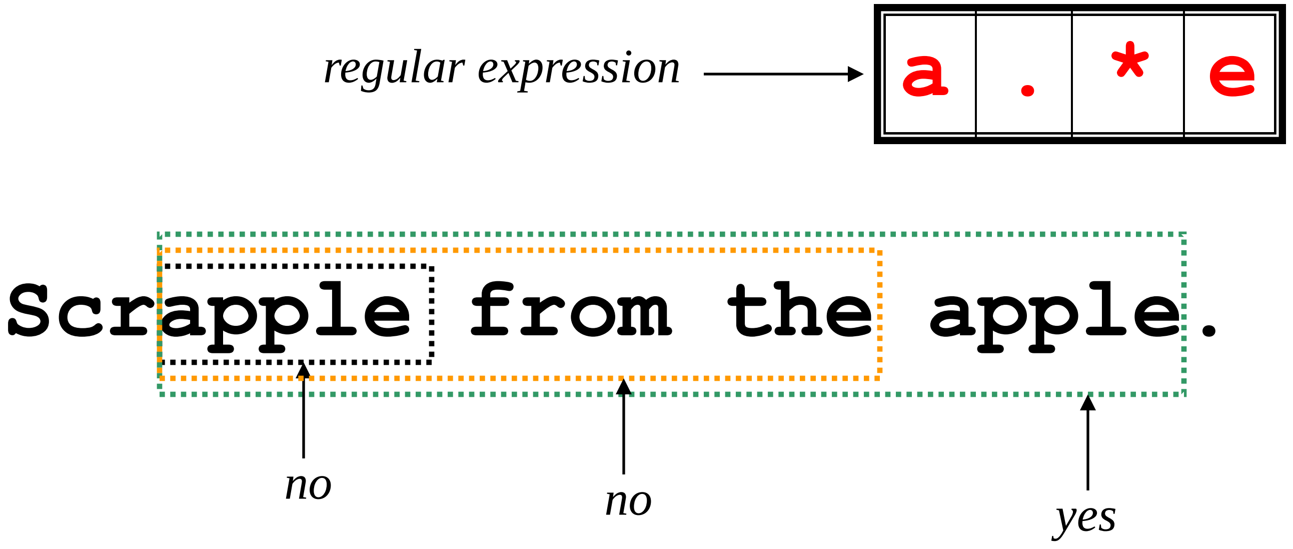
- The * is used to define **zero or more** occurrences of the *single* regular expression preceding it.


regular expression
y a * y
I got mail, yaaaaaaaaaay!
match


regular expression
y a * y
I got mail, yyaaa!
match

Match length

- A match will be the longest string that satisfies the regular expression.



Repetition Ranges

- Ranges can also be specified
  - `{ }` notation can specify a range of repetitions for the immediately preceding regex
  - `{`***n***`}` means exactly *n* occurrences
  - `{`***n***`,}` means at least *n* occurrences
  - `{`***n***`,`***m***`}` means at least *n* occurrences but no more than *m* occurrences
- Example:
  - `.{0,}` same as `.*`
  - **`a{2,}`** same as **`aaa*`**

Subexpressions

- For grouping part of an expression so that `*` or `{ }` applies to more than just the previous character, use `( )` notation
- Subexpresssions 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`**

- Syntax

  grep [-hilnv] [-e expression] [filename]
  - **-h** Do not display filenames
  - **-i** Ignore case
  - **-l** List only filenames containing matching lines
  - **-n** Precede each matching line with its line number
  - **-v** Negate matches
  - **-x** Match whole line only (*fgrep* only)
  - **-e** *expression* Specify expression as option
  - **-f** *filename* Take the regular expression (egrep) or a list of strings (fgrep) from *filename*

grep

- Example:

```
$ ls | grep -e ‘ug*’
ug1
ug2
ug3
ug4
```

Escaping Special Characters

- The shell interprets `*` and `.` as special characters to **grep**
- To get literal characters, *escape* the character with a \ (backslash)
- For searching the character sequence **`a*b*`**
  - This will match zero or more ‘a’s followed by zero or more ‘b’s, *not the desired*
  - **`a\*b\*`** will fix this - now the asterisks are treated as regular characters