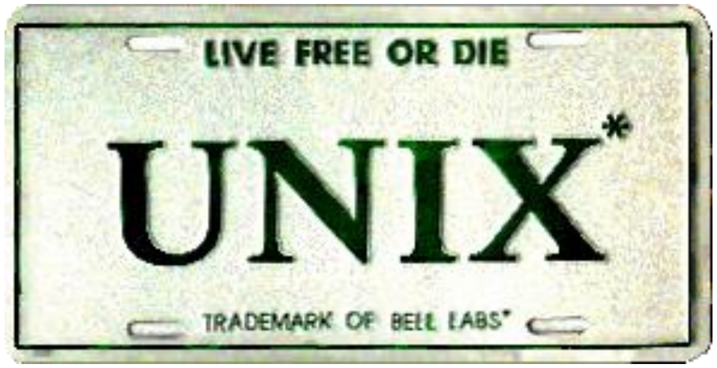


UNIX for Programmers and Users

"UNIX for Programmers and Users"
Third Edition, Prentice-Hall, GRAHAM GLASS, KING ABLES

sort

- Syntax: *sort [-rn] [filename(s)]*

-*r*	Sort in reverse order
-*n*	Numeric order

uniq: list UNIQue items

- Remove or report adjacent duplicate lines
- Syntax: *uniq [-cdu] [input-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*

  Ex.

```
cat f1 | uniq
cat f1 | sort | uniq
```

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

Ex.

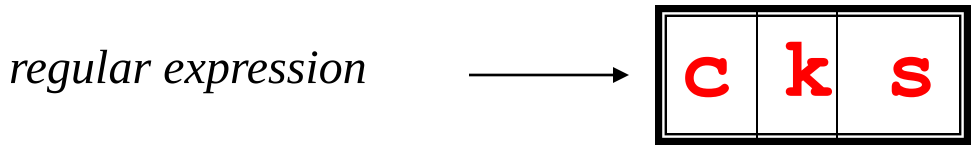
```
cat f1 | tr s r
```

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** and **emacs**
  - **awk** and **Python**
  - **grep**
  - **compilers**



```
$ echo "UNIX Tools rocks" | grep "cks"
```

```
UNIX Tools rocks.
```

match

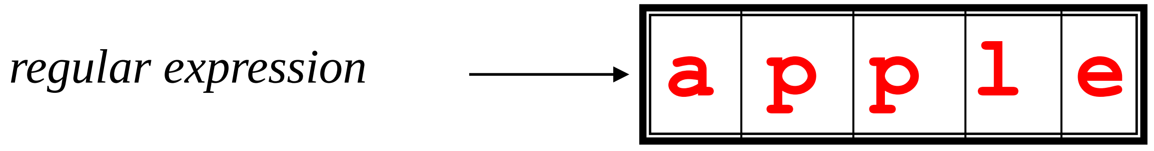
```
$ echo "UNIX Tools okay" | grep "cks"
```

```
UNIX Tools okay.
```

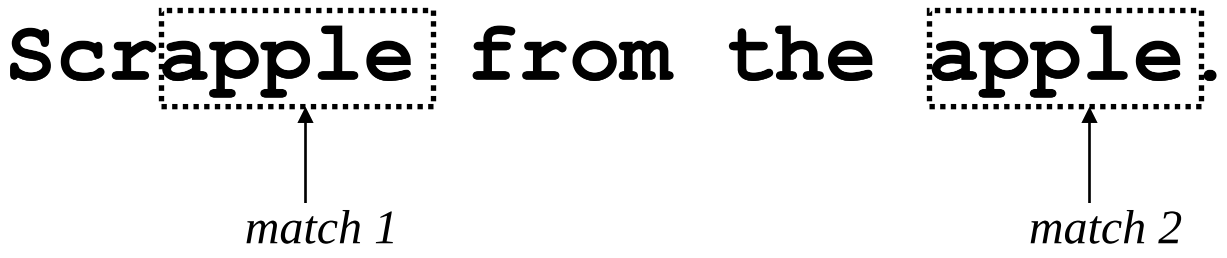
no match

Regular Expressions

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



```
$ echo "Scapple from the apple" | grep "apple"
```

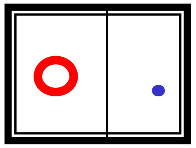


```
Scrapple from the apple.
```

Regular Expressions

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

regular expression →



```
$ echo "Suggesion for you - work hard" | grep "o."
```

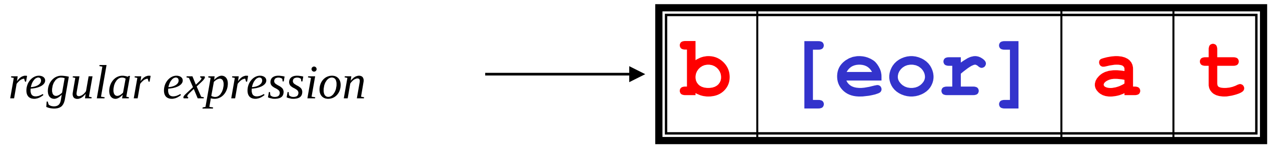
```
Suggestion for you - work hard.
```

match 1

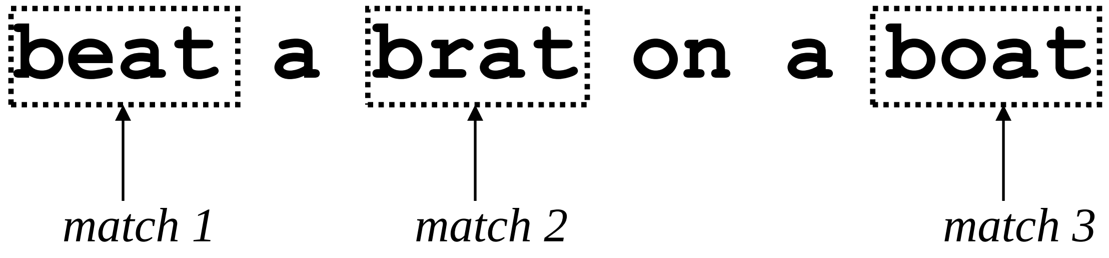
match 2

Character Classes

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

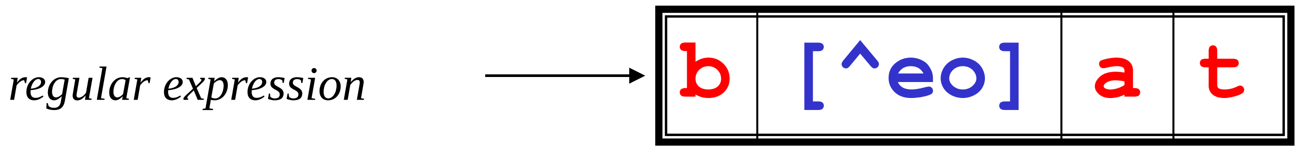


```
$ echo "beat a brat on a boat" | grep "b[eor]at"
```

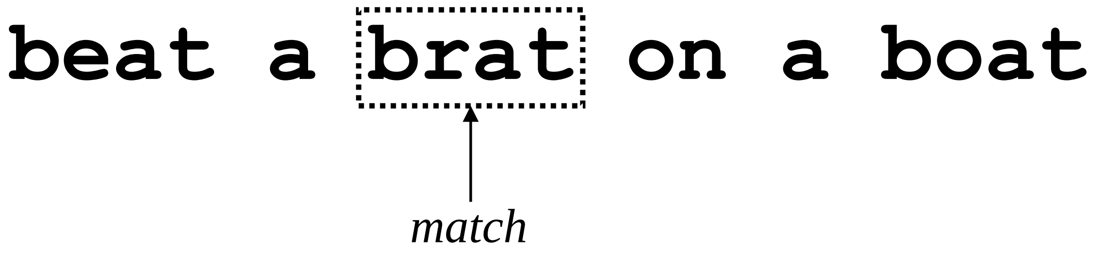


Negated Character Classes

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



```
$ echo "beat a brat on a boat" | grep "b[^eo]at"
```



More About Character Classes

  - `[aeiou]` will match any of the characters `a`, `e`, `i`, `o`, or `u`
  - `[kK]orn` will match `korn` or `Korn`

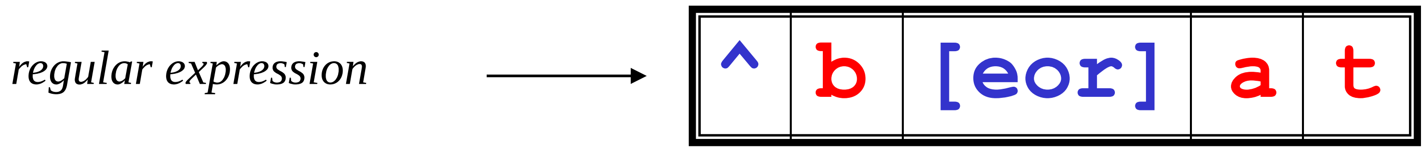
- Ranges can also be specified in character classes
  - `[1-9]` is the same as `[123456789]`
  - `[a-e]` is equivalent to `[abcde]`
  - Multiple ranges can be combined also
    - `[a-e1-9]` is equivalent to `[abcde123456789]`
  - Note that the `-` character has a special meaning in a character class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

Named Character Classes

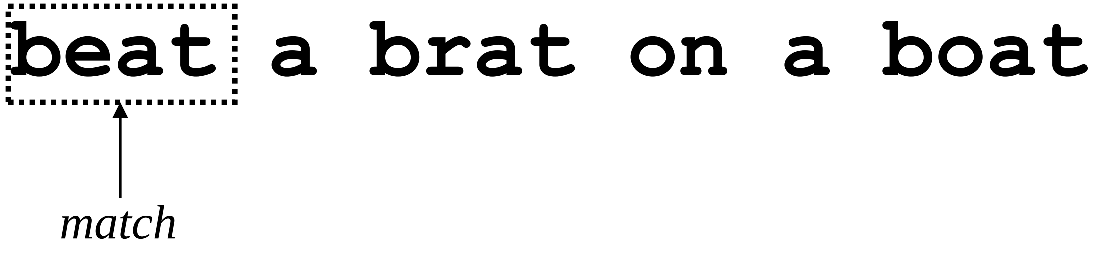
- Commonly used character classes can be referred to by name (*alpha*, *lower*, *upper*, *alnum*, *digit*, *punct*, *cntrl*)
- Syntax `[:`***name***`:]`
  - `[a-zA-Z]` `[[:alpha:]]`
  - `[a-zA-Z0-9]` `[[:alnum:]]`
  - `[45a-z]` `[45[:lower:]]`
- Important for portability across languages

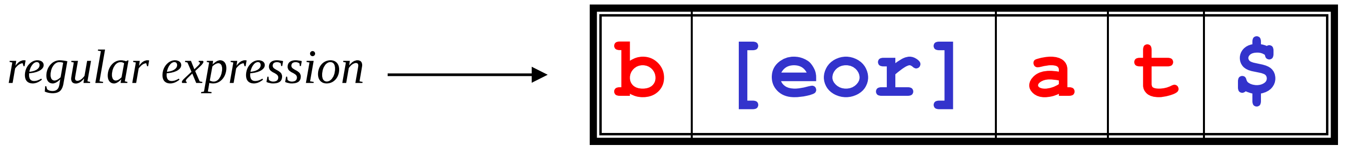
Anchors

- 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

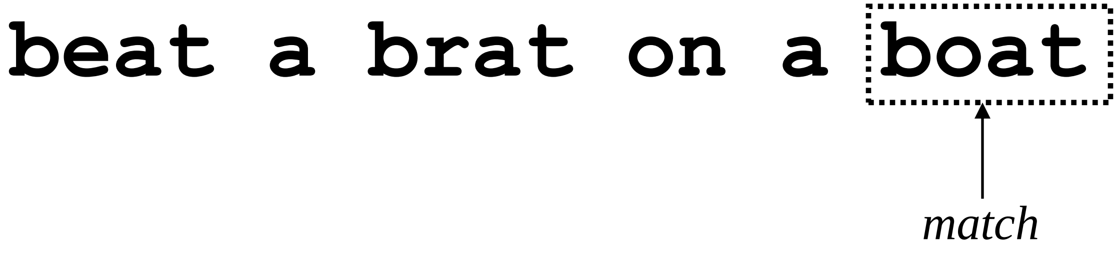


```
$ echo "beat a brat on a boat" | grep "^b[eor]at"
```



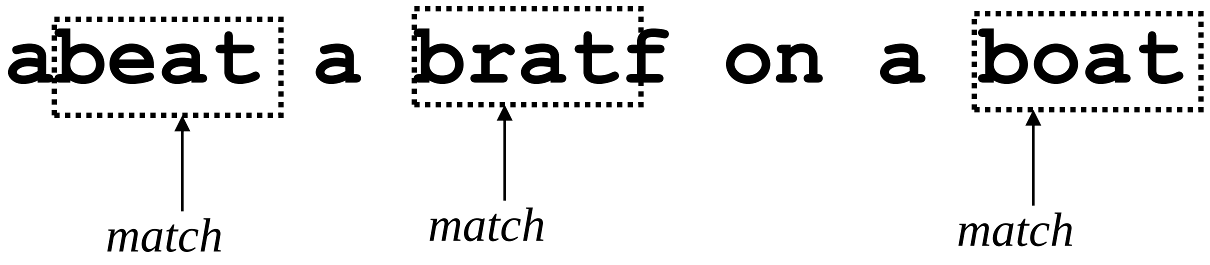


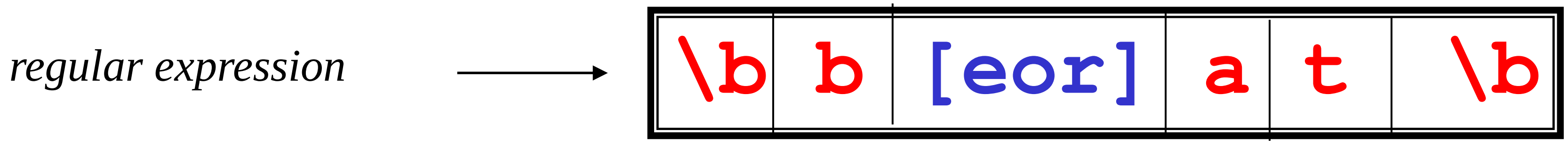
```
$ echo "beat a brat on a boat" | grep "b[eor]at$"
```



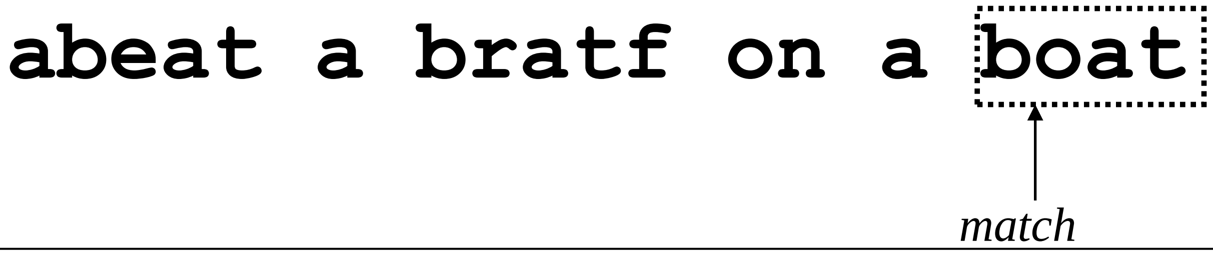
Mathching Exact World

```
$ echo "abeat a bratf on a boat" | grep "b[eor]at"
```



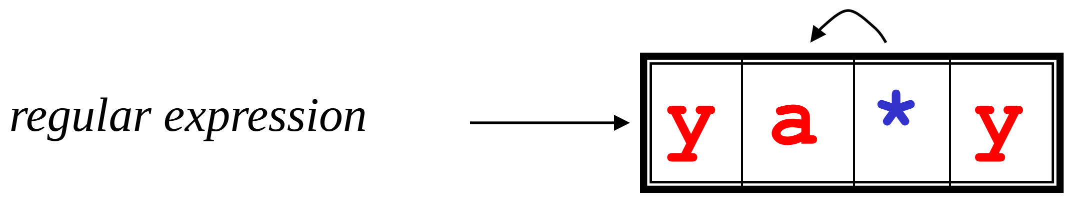


```
$ echo "abeat a bratf on a boat" | grep "\bb[eor]at\b"
```

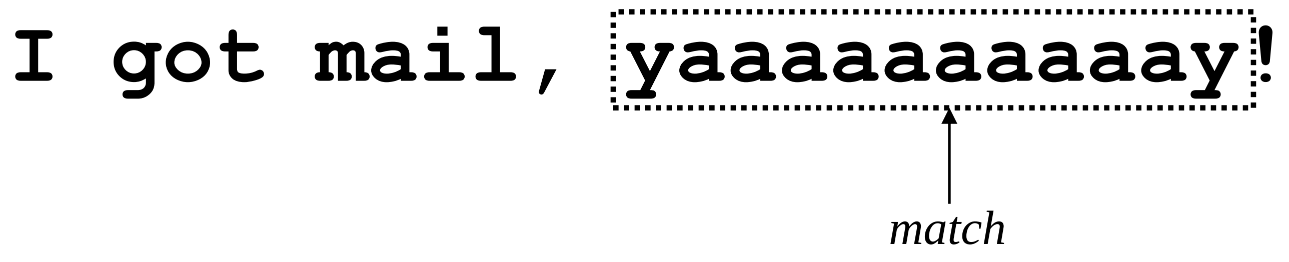


Repetition

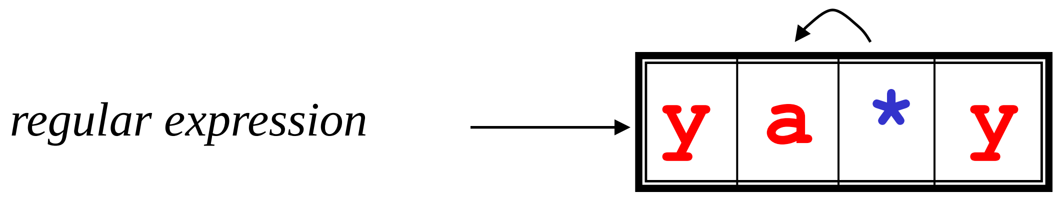
- The * is used to define **zero or more** occurrences of the *single* regular expression preceding it.



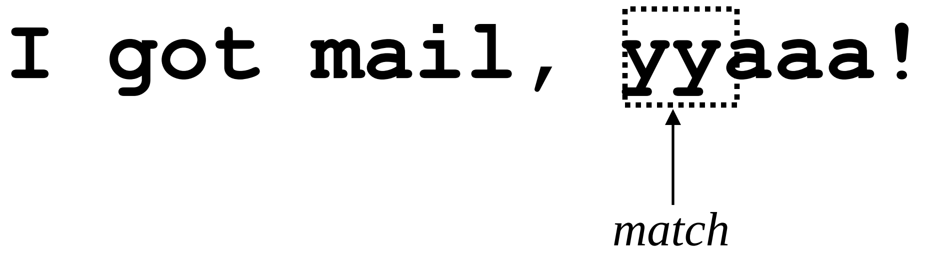
```
$ echo "I got mail, yaaaaaaaaaaay!" | grep "ya*y"
```



```
I got mail, yaaaaaaaaaaay!
```



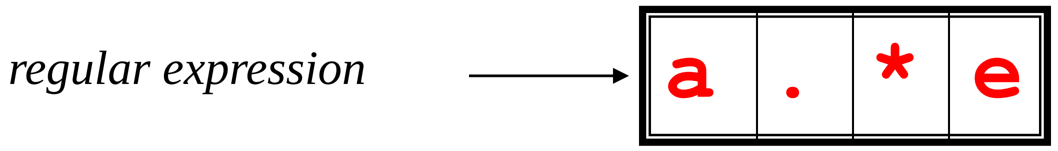
```
$ echo "I got mail, yyaaa!" | grep "ya*y"
```



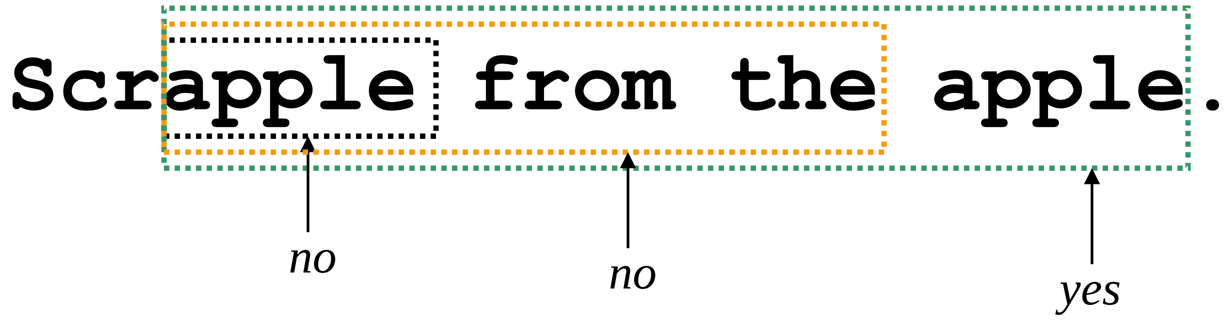
```
I got mail, yyaaa!
```

Match length

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



```
$ echo "Scrapple from the apple" | grep "a.*e"
```



Repetition Ranges

egrep (or) grep -E

- 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*`

Ex.

```
echo aaa aa a | grep -E "a{2,}"
```

Repetition Ranges

egrep (or) grep -E

```
$ echo "a aa aaa aaa aaaaa aaaa" | egrep "a{2}"

$ echo "a aa aaa aaa aaaaa aaaa" | egrep "a{3}"

$ echo "a aa aaa aaa aaaaa aaaa" | egrep "a{2,}"

$ echo "a aa aaa aaa aaaaa aaaa" | egrep "a{2,3}"
```

Subexpressions

With grep -E (or) egrep

- 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`

Ex.

```
egrep "aa(aa)*" f1
  echo aa aaa aaaa aaaaa aaaaaa | egrep "aa(aa)*"
```

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 [-hilnvE] [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
– **-E**	*expression* - Specify expression

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