Looking for fake words

```
Subject: Re: 90 days
Subject: [SPAM:XXXXXXXXX]
Subject: Fancy rep1!c@ted watches
```

Look for any digit surrounded by letters–

```
[abcdefghijklmnopqrstuvwxyz][0123456789][abcdefghijklmnopqrstuvwxyz]
```

Look for any punctuation surrounded by letters–

```
[abcdefghijklmnopqrstuvwxyz][.;:!?][abcdefghijklmnopqrstuvwxyz]
```

Do you see any problems with these patterns?

Named character classes

`[:alpha:]` stands for any letter

`[:digit:]` any digit - 0 1 2 3 4 5 6 7 8 9

`[:punct:]` stands for any punctuation mark

`[:alnum:]` means any letter or digit

Return to the search for any digit surrounded by letters.

```
[[:alpha:]][[:digit:]][[:alpha:]]
```

Look for any punctuation surrounded by letters.

```
[[:alpha:]][:;,!?$@%][[:alpha:]]
```

Why did I choose to not use the named characeter class `[:punct:]`?

We still have a problem with not being able to pick up a word like, `rep1!cated`.

Meta characters

These are characters that have special meaning. Here are a few:

`.` any character
`*` the previous character may appear 0 or more times
`?` the previous character may appear 0 or 1 time
`+` the previous character may appear 1 or more times
`^` anchor the search to the beginning of the string
`$` anchor the search to the end of the string
`(  )` treat what is between the parentheses as a single entity

Return to the search for any digit surrounded by letters.

```
[[:alpha:]][[:digit:];:,!@#$%&*?]+[[:alpha:]]
```

Notice the punctuation and the digits are within the same character class.
Notice that there must be at least one occurrence of punctuation mark or digit between two letters for the search to find a match.

Some handy R functions

```
countynames
[1] "De Witt County"            ...
[4] "St John the Baptist Parish"

# The strsplit function splits of each character string at the blanks
words = strsplit(countynames, split=" ")

words
[[1]]
[1] "De"     "Witt"   "County"  ...
[[4]]
[1] "St"      "John"    "the"     "Baptist" "Parish"

sapply(words, function(x) {paste(x[-length(x)], collapse=" ")})
[1] "De Witt"              "Lac qui Parle"
[3] "Lewis and Clark"      "St John the Baptist"
```

Some handy R functions: gsub

gsub(pattern, replacement, search string)

gsub stands for global substitution.
If the pattern is found in the search string then it is replaced with the replacement string.
The **sub** function, is similar to **gsub**, but it only makes the replacement for the first pattern found in the search string, rather than for all occurrences of the pattern.

```
countynames
[1] "De Witt County"            ...
[4] "St John the Baptist Parish"

gsub("( County)|( Parish)", "", countynames)
[1] "De Witt"            "Lac qui Parle"
[3] "Lewis and Clark"    "St John the Baptist"
```

Notice the blank that precedes County and Parish, why is it there?

What do you think the parentheses are doing here?

Some handy R functions: grep

grep(pattern, search string)
Some say that **grep** stands for grab regular expression. Recall, the logical vector that we want to create to indicate if the subject line of an email contains fake works.

```
subjects
[1] "Subject: Re: 90 days"
[2] "Subject: [SPAM:XXXXXXX]"
[3] "Subject: Fancy rep1!c@ted watches"
grep("[[:alpha:]][!,;:.?][[:alpha:]]", subjects)
[1] 2
grep("[[:alpha:]][[:punct:]][[:alpha:]]", subjects)
[1] 2 3

grep("[[:alpha:]][!,;:.?[:digit:]][[:alpha:]]", subjects)
[1] 2
grep("[[:alpha:]][!,;:.?[:digit:]]+[[:alpha:]]", subjects)
[1] 2 3
```

Why does the first pattern not match the third string, but the second pattern does?
Why does the third pattern not match the third string, but the fourth pattern does?

Some handy R functions: regexpr

The **regexpr** provides more information about the location of the pattern in the search string. It returns the position in the character string of the start of the match. It also returns as an attribute of the return vector, the length of the matching substring. If not mathc is found, it returns -1.

```
subjects
[1] "Subject: Re: 90 days"
[2] "Subject: [SPAM:XXXXXXX]"
[3] "Subject: Fancy rep1!c@ted watches"

regexpr("[[:alpha:]][[:punct:]][[:alpha:]]", subjects)
[1] -1 14 21
attr(,"match.length")
[1] -1  3  3

regexpr("[[:alpha:]][!,;:.?[:digit:]]+[[:alpha:]]", subjects)
[1] -1 14 18
attr(,"match.length")
[1] -1  3  4
```

Escaping Metacharacters

Sometimes, we might want to search for a `[` or a `.` or a `?` so how do we tell the regular expression language that the character is not a meta-character but the actual character itself?

We need to “escape” it’s meaning by putting a backslash in front of it `\[`
Since R also uses the backslash as an escape character, we need to put two backslashes, the extra one is to escape the backslash so it is passed on to the regular expression language.

```
 web
[1] "169.237.46.168 - - [26/Jan/2004:10:47:58 -0800] \"GET  /stat141/Wint
> regexpr("[.*]", web)
[1] 4
attr(,"match.length")
[1] 1
> regexpr("\\[.*\\]", web)
[1] 20
attr(,"match.length")
[1] 28
```

How is the `.` used in the first pattern? Why is there a match at character 4 in the string?
Why is there a backslash before `"GET`?

Here is how we might use the information returned from **regexpr** to pull out a substring from the search string.

```
 web
[1] "169.237.46.168 - - [26/Jan/2004:10:47:58 -0800] \"GET
      /stat141/Winter04 HTTP/1.1\" 301 308"

loc = regexpr("\\[.*\\]", web)
substring(web, loc+1, loc + attr(loc, "match.length")-8)
[1] "26/Jan/2004:10:47:58"
```

Why do we subtract 8 from the value of match.length?

Variables

It is possible to reference a matched pattern.

```
> web
[1] "169.237.46.168 - - [26/Jan/2004:10:47:58 -0800] \"GET
     /stat141/Winter04 HTTP/1.1\" 301 308"

gsub('(.*) - - \\[(.*) [-+][0-9]{4}\\] "(GET|POST).*',
     '\\1, \\2, \\3', web)
[1] "169.237.46.168, 26/Jan/2004:10:47:58, GET"
```

Each parenthetical pattern is treated as a variable. The first is `\\1`
What is this **gsub** doing?

Create Regular Expressions for these 3 examples

- Find a time in the format “12:30 pm” or “9:15 AM” or “08:20 am”

String	12:30 pm	2:15 AM	312:23 pm	1:00 american	08:20 am
	Yes	Yes	No	No	Yes

- Find the dollars and cents where the cents are optional.

String	$12	$.99	$0.75	$12.23	$1.234	$1010.01	$06.20
	Yes	Yes	Yes	Yes	No	Yes	No

- Find double words

String	Blythe the great.	What is an answer?	It is the the best!	Please! Help help!
	No	No	Yes	Yes