Regular expressions

"regex": a language for specifying text search patterns

basic

Without any special characters, the pattern is a simple case-sensitive match

`hello`

```
 matches       | does not match
---------------+----------------
 hello world   | Hello
 shelloser     | hell-o
 goodbye hello |
```

special characters

`.|\*+?{}()[]^$`

Note that within any container (`{}` , `()` , `[]`),
there may be different sets of special characters.

Use `\` to "escape" these if you mean to use them literally.

e.g. `\?` matches the string "?"

quantifiers

- `?` - zero or one times
- `*` - zero or more times
- `+` - one or more times
- `{n}` - exactly n times
- `{n,m}` - n to m times
- `{n,}` - at least n times
- `{,m}` - at most m times

`a+r+g` matches "arg", "aaarrrrrg", etc.

disjunctions

`Alice|Bob` matches "Alice" or "Bob".

simple groups

`(super|bat)man` matches "superman" or "batman".

exercise (1)

Write an expression that matches simple street addresses, e.g.

- "1600 Pennsylvania Avenue"
- "401 Chapel Drive"
- "3624 Hillsborough Road"
- ...

Use only:

- literal characters
- quantifiers
- `|`, `(`, `)`

sets

The special characters `[` and `]` delimit a set of characters.

`[hymb]ello` can be considered a shortcut for `(h|y|m|b)ello`

```
 matches         | does not match
-----------------+-----------------
 hello           | Hello
 yellow          | below
 mello           |
```

Within sets, the only special characters are `-^]`

exercise (2)

Write an expression that matches simple street addresses, e.g.

- "1600 Pennsylvania Avenue"
- "401 Chapel Drive"
- "3624 Hillsborough Road"
- ...

Use only:

- literal characters
- quantifiers
- `|`, `(`, `)`, `[`, `]`

ranges

Within brackets, `-` is a special character, allowing ranges of characters:

- lower-case letters e.g. `[a-f]` ↔ `[abcdef]`
- upper-case letters e.g. `[R-U]` ↔ `[RSTU]`
- digits e.g. `[2-7]` ↔ `[234567]`

`[c-gxyz]` ↔ `[cdefgxyz]`

negation

If the first character after `[` is `^`, the set is negated.

`[^h]` means "any character other than `h`".

`[^h]ello`

```
 matches         | does not match
-----------------+-----------------
 Hello           | hello
 yellow          |
 mello           |
```

exercise (3)

Write an expression that matches simple street addresses, e.g.

- "1600 Pennsylvania Avenue"
- "401 Chapel Drive"
- "3624 Hillsborough Road"
- ...

Use only:

- literal characters
- quantifiers
- `|`, `(`, `)`, `[`, `]`, `-`

shortcuts

- `\d` - digit; equivalent to `[0–9]` for ASCII (plus other digits for Unicode)
- `\D` - non-digit; equivalent `[^0–9]` for ASCII
- `\s` - whitespace character, including space, `\t`, `\n`, ...
- `\S` - non-whitespace character
- `\w` - alphanumeric character; equivalent to `[a–zA–Z0–9_]` for ASCII
- `\W` - non-alphanumeric character
- `\b` - word boundary; matches the empty string, but only at the beginning or end of a word
- `\B` - non-word boundary

exercise (4)

Write an expression that matches simple street addresses, e.g.

- "1600 Pennsylvania Avenue"
- "401 Chapel Drive"
- "3624 Hillsborough Road"
- ...

Use only:

- literal characters
- quantifiers
- `|`, `(`, `)`, `[`, `]`, `-`
- "shortcuts"

wildcards

`.` matches any single character

`p.t`

```
 matches        | does not match
----------------+----------------
 pat            | pt
 pet            | pout
 pit            |
 pot            |
 put            |
```

anchors

- `^` matches the beginning of a string or line
- `$` matches the end of a string or line
- `\b` / `\B` could also be considered an anchor...

greed

By default regex patterns match "greedily", i.e. as much text as possible:

- `\w+` matches two substrings in "hello world"

Many implementations use the non-greedy modifier `?` that can follow any quantifier:

- `\w+?` matches 10 substrings in "hello world"

group extensions

- `(?:...)` - a non-capturing group
- `(?=...)` - lookahead
  - matches the empty string if followed by a match for `...`
- `(?!...)` - negative lookahead
  - matches the empty string if not followed by a match for `...`
- `(?<=...)` - lookbehind
  - matches the empty string if the preceded by a match for `...`
- `(?<!...)` - negative lookbehind
  - matches the empty string if not preceded by a match for `...`

Note: regex implementations vary a bit in support for these.

lookarounds examples

Find the last names of famous Patricks:

> Patrick Wang
> Patrick Swayze
> Jeremy Wang

```
(?<=Patrick )\w+
```

group references

- `\1`, `\2`, etc. - matches the contents of the (capturing) group of the same number
- `(?P<name>)` - identical to a simple capturing group except that a name is associated with the group
- `(?P=quote)` - a backreference to a named group

Note: regex implementations vary a bit in support and syntax for these.

example

> The family of Dashwood had long been settled in Sussex. Their estate was large, and their residence was at Norland Park, in the centre of their property, where, for many generations, they had lived in so respectable a manner as to engage the general good opinion of their surrounding acquaintance.

Match all words that do not include repeated characters (e.g. "good").

example (solution)

```
(.)\1
```

```
\b\w*(.)\1\w*\b
```

```
\b((?!(.)\2)\w)+\b
```

exercise (5)

Write an expression that matches only the middle bit of simple street addresses, e.g.

- "Pennsylvania" in "1600 Pennsylvania Avenue"
- "Chapel" in "401 Chapel Drive"
- "Hillsborough" in "3624 Hillsborough Road"
- ...

references

- https://web.stanford.edu/~jurafsky/slp3/2.pdf, section 2.1
- https://docs.python.org/3/howto/regex.html#regex-howto
- https://docs.python.org/3/library/re.html

sandboxes

- https://regexr.com/
- https://regex101.com/

practice

- http://alf.nu/RegexGolf