Regular Expressions:

List of all metacharacters: `. ^ $ * + ? ] [ \ | ( ) { }`

Operator	Description
`.`	Matches any character except \n
`\`	Escapes metacharacters
`|`	Matches expression on either side of expression; has lowest priority of any operator
`\d, \w, \s`	Predefined character group of digits (0-9), alphanumerics (a-z, A-Z, 0-9, and underscore), or whitespace, respectively
`\D, \W, \S`	Inverse sets of \d, \w, \s, respectively
`*`	Matches preceding character/group zero or more times
`?`	Matches preceding character/group zero or one times
`+`	Matches preceding character/group one or more times
`*?, +?`	Applies non-greedy matching to * and +, respectively
`{m}`	Matches preceding character/group exactly m times
`{m, n}`	Matches preceding character/group at least m times and at most n times; if either m or n are omitted, set lower/upper bounds to 0 and ∞, respectively
`^, $`	Matches the beginning and end of the line, respectively
`[ ]`	Matching group used to match any of the specified characters or range (e.g. [abcde]) [a-e])
`( )`	Capturing group used to create a sub-expression
`[^ ]`	Invert matching group; e.g. [^a-c] matches all characters except a, b, c

Regex String Matching:

Function	Description
`re.match(pattern, string)`	Returns a match if zero or more characters at beginning of `string` matches `pattern`, else None
`re.search(pattern, string)`	Returns a match if zero or more characters anywhere in `string` matches `pattern`, else None
`re.findall(pattern, string)`	Returns a list of all non-overlapping matches of `pattern` in `string` (if none, returns empty list)
`re.sub(pattern, repl, string)`	Returns `string` after replacing all occurrences of `pattern` with `repl`

Data 100 Regular Expressions

(Spring 2022)
Here's a complete list of metacharacters:

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

Some reminders on what each can do (this is not exhaustive):

`"^"` matches the position at the beginning of string (unless used for negation `"[^]"`)

`"$"` matches the position at the end of string character.

`"?"` match preceding literal or sub-expression 0 or 1 times.

`"+"` match preceding literal or sub-expression *one* or more times.

`"*"` match preceding literal or sub-expression *zero* or more times

`"."` match any character except new line.

`"[ ]"` match any one of the characters inside, accepts a range, e.g., `"[a-c]"`.

`"( )"` used to create a sub-expression

`"\d"` match any *digit* character. `"\D"` is the complement.

`"\w"` match any *word* character (letters, digits, underscore). `"\W"` is the complement.

`"\s"` match any *whitespace* character including tabs and newlines. `\S` is the complement.

`"*?"` Non-greedy version of *. Not fully discussed in class.

`"\b"` match boundary between words. Not discussed in class.

`"+?"` Non-greedy version of +. Not discussed in class.

`"{m,n}"` The preceding element or subexpression must occur between m and n times, inclusive.

Some useful `re` package functions:

`re.split(pattern, string)` split the `string` at substrings that match the `pattern`. Returns a list.

`re.sub(pattern, replace, string)` apply the `pattern` to `string` replacing matching substrings with `replace`. Returns a string.

`re.findall(pattern, string)` Returns a list of all matches for the given `pattern` in the `string`.