```python
# Regular Expressions
import re
import pandas as pd

text = """
Long ago, I travelled to the far west, seeking my fortune. I found
frosty mountains, arid deserts, lush oases, and a huge ocean.
At times, I was gripped by despair, and at other times filled with Joy.

- Anonymous, 1865
"""

with open("data/filenames.txt") as f:
    filenames = f.readlines()
print(filenames[0])
```

Guide (works in both python and R)

- Letters, Numbers match themselves
- '.' matches one of anything
- '+' means one or more of the preceeding
- '*' means 0 or more of the preceding
- '?' matches 0 or 1 occurrences of the previous pattern.
- [] groups things ([A-Z]+ matches a sequence of one or more capital letters); [^...] matches anything *not* in the range.
- '\w' matches "word" characters ('[a-zA-Z0-9_]')
- '\W' matches non-word characters
- '\b' matches boundaries (end or start of string)
- '{5}" matches 5 times
- '{3,5}' matches 3, 4 or 5 occurrences.
- '{3,}' matches 3 or more occurrences
- '\s' matches whitespace
- '\S' matches non-whitespace
- '^....' matches at the start of a line
- '...$' matches at the end of a line
- '(a|b)' matches a or b.
- Use backslash to escape.

Key functions

- `match` finds matches *at the start of the string*; returns None if it doesn't find one, otherwise returns match object.
- `search` finds matches; returns None if it doesn't find one, otherwise returns first match object
- `findall` returns a list of all matches (not match objects)
- `finditer` iterates over matches

Match objects

- if `m` is a match object, then
  1. `m[0]` is the match
  2. `m[2]`, `m[3]` and so on are the subgroup matches
  3. `m.span(n)` is (start, stop) for match n.
  4. `m.start(n)` and `m.end(n)` are the start and end of match n.
  5. `m.string` is the string being matched against

Looking for explicit strings

```
if re.search(r"travel", text):
    print("Yes")
else:
    print("No")

if re.match(r"travel", text):
    print("Yes")
else:
    print("No")
```

Some fancier examples

```
# All the words
all_words = re.findall(r"\b[a-zA-Z]+\b", text)
all_words[0:5]

# words (allowing numbers and underline) but lower case
re.findall(r"\b\w+\b", text.lower())[0:5]

# numbers
re.findall(r"\b\d+\b", text)

regular = re.search(r'[A-Z][a-z]+',text)
short = re.search(r'[A-Z][a-z]?',text)

#Compare these
plus = re.findall(r'[A-Z][a-z ]+',text)
plusq = re.findall(r'[A-Z][a-z ]+?',text)

# Finding capitalized words
re.findall(r"\b[A-Z][a-z]*\b", text)

# Problem: Find all sentences (Start with capital letter, end with period. Remember to use
```

An example

```
with open("data/filenames.txt","r") as f:
    filenames = f.readlines()
```

```
print(filenames[0])
filenames = [x.strip() for x in filenames] #get rid of the newlines

# Using alternation to select qmd or Rmd files
selected = [x for x in filenames if re.match(r".*\.(qmd|Rmd)",x)]
rejected = [x for x in filenames if not re.match(r".*\.(qmd|Rmd)",x)]

# Using grouping to extract netid
matches = [re.search(r"_([a-z]{3}[0-9]{5})_",x) for x in selected]
[x[1] for x in matches][0:5]

filenames = pd.read_csv("data/filenames.txt",names=["Name"])

filenames['Name'].map(lambda x: re.search(r"_([a-z]{3}[0-9]{5})_",x)[1])
filenames = filenames.assign(netid = filenames['Name'].map(lambda x: re.search(r"_([a-z]{3}
 )
filenames = filenames.assign(extension = filenames['Name'].map(lambda x: re.search(r".*\.(q
```

Adding `(?P<name>...)` names the submatch. You can then extract or refer to the submatch by name.

```
m = re.search(r"(?P<found>[a-z]{3})","abcdefghij")
print(m[0],m.group(1),m.group('found'))
```

The `.str.extract` method is a powerful way to pick apart a string into columns in a pandas dataframe. It combines the operations above into a single operation. Combining it with named submatches gives names to the new columns.

```
filenames = pd.read_csv("data/filenames.txt",names=["Name"])
filenames=filenames['Name'].str.extract(r"(?P<name>.*_(?P<netid>[a-z]{3}[0-9]{5})_.*\.(?P<ex
```

There are many other useful operations available with the pandas str library.

- `str.split`
- `str.replace`
- `str.cat` (joins strings together with argument `sep=`)