Regular Expressions

A regular expression is a pattern that matches text.

The `re` Library

The `re` library in Python provides regular expression functionality.

We have to develop two parts, and it's good to keep them separate in our minds:

- The Python part
- The regular expression part

The Python Part

We have two ways of handling regular expressions in Python:

1. We can call methods that are part of the `re` library that include a regular expression pattern.
2. We can compile a pattern and call methods against the resulting object.

We can use either, but if we compile patterns, we have access to the objects later if we need them again.

Python `re` Methods

- `compile` – This compiles a regular expression pattern into a regular expression object. We can then call `match`, `findall`, `search`, and `sub` against this object.
    - `pattern = re.compile(r'#.*')`
- `match` – This matches 0 or more characters at the beginning of a string (i.e. not in the centre of a string). Returns a match object.
    - `match = pattern.match(stringtomatch)`

- `search` – This scans through a string to find the first match that occurs. Returns a match object.
    - `match = pattern.search(stringtomatch)`
- `findall` – This returns all non-overlapping matches as a list of strings in the order in which they were found.
    - `matches = pattern.findall(string)`
- `sub` – This returns the string obtained by replacing occurrences of a pattern with the replacement. If no matches were found, the original string is returned.
    - `line = pattern.sub('', line)`

More methods and details are available in the python documentation.

The Regex Part

There is a good tutorial here:

- https://docs.python.org/3.4/howto/regex.html#regex-howto

Regex is useful when we add meta-characters to describe wildcards, repetition, and iteration.

- The `.` character matches any character other than the newline character. Using `findall` with this will return each line.
- The `*` character matches 0 or more occurrences of the regex that precedes it.
- The `+` character matches 1 or more occurrences of the regex that precedes it.

Note we need to use brackets for grouping:

- `python*` matches 'pytho', 'python' and 'pythonnnnnn', as the `*` is applied to just the `n`
- `(python)*` matches '', 'python', and 'pythonpythonpython', as the `*` is applied to the whole expression contained in the brackets.

Example

Say we want to match Python comments:

```
pattern = re.compile('#.*')
matches = pattern.findall(code)

for match in matches:
    print(match)
```

If code is a string that contains code with comments, the code above will output all comments in that code.

Grouping

We can also group characters.

- `[abc]` will match `a` or `b` or `c`. `[Pp]ython` will match 'python' or 'Python'.

Shorthands

- `\w` – matches all word characters
- `\s` – matches all whitespace
- `\d` – matches digits
- `\b` – matches word boundaries

Example 2

We want to match all capitalised words in a string (except one-letter words e.g. 'I').

1. We start by matching word boundaries: `'\b\b'`
2. Next match a capital letter: `'\b[A-Z]\b'`
3. Next 1 or more lowercase letters: `'\b[A-Z][a-z]+\b'`

Then we compile this and use the resulting object to match the pattern:

```
pattern = re.compile(r'\b[A-Z][a-z]+\b')
matches = pattern.findall(string)

for match in matches:
    print(match)
```

This pattern will match "This", "Name", and "Placename" in this example sentence:

- This A123 with Name & Placename

Specifying Multiples

To specify 3 digits, for example, you can write `\d{3}`. To specify an upper-bound, use e.g. `\d{3,6}` to match sequences of between 3 and 6 digits.

Example: Phone Numbers

We want to match the following options:

- (086)12345
- (021) 12345
- +353(21)12345
- 00353 12 12345

To match the first two, we can use `"\(\d{3}\)\s*\d{5,7}"`. This matches:

1. An open bracket
    - Note that we have to escape the `(` and `)` characters in the regex.
2. Followed by a sequence of 3 digits
3. Followed by a close bracket
4. Followed by 0 or more whitespace characters
5. Followed by a sequence of 5, 6, or 7 digits.

To match the third, we can use `"\+\d{3}\s*\(\d{2}\)\s*\d{5,7}"`.

[didn't take down the last one]