LECTURE 28

REGULAR EXPRESSIONS

MCS 260 Fall 2020

Emily Dumas

REMINDERS

- If you haven't started Project 3, you are behind!
- Worksheet 10 available
- Quiz 10 coming Thursday, due Nov 2
- Nov 3: All UIC courses canceled (Election day)
- Nov 5: Extra TA office hours instead of discussions

LOOSE END: RECURSION PROS AND CONS

Often can solve a problem with recursion or with loops (an **iterative** solution). Why use recursion?

Pros:

- Short code
- Clear code

Unclear:

- Speed

Cons:

- Uses more memory

REGULAR EXPRESSIONS

Today we'll learn about the module `re` in Python, which supports a text searching language known as **regular expressions** or **regexes**.

Some of its key functions include:

- Searching for text matching a pattern
- Replacing text matching a pattern

MINIMAL EXAMPLE

Regexes are a mini programming language for specifying patterns of text. Dialects of regex are supported in many programming languages. We'll cover the Python dialect.

Simplest usage: Find and replace a substring.

```
import re
s = "Avocado is usually considered a vegetable."
print(re.sub("vegetable","fruit",s))
```

```
re.sub(pattern, replacement, string)
```

The first argument of `re.sub` is a **pattern**.

Unless it contains characters with special meaning in a regex pattern, the pattern just matches substrings equal to the pattern.

- `"vegetable"` matches the string `"vegetable"`
- `"foo"` matches the string `"foo"`

RAW STRINGS

Recall that backslash `\` in a string starts an escape sequence in Python, and `\\` represents a single backslash character in the string. If your string contains a lot of backslashes, you may want to disable escape sequences.

You can do so by putting the letter `r` immediately before the quotation mark(s). This is known as a **raw string**. In a raw string, a single \ represents the \ character.

SPECIAL CHARACTERS IN PATTERNS

- `.` — matches any character except newline
- `\s` — matches any whitespace character
- `\d` — matches a decimal digit
- `+` — previous item must repeat 1 or more times
- `*` — previous item must repeat 0 or more times
- `?` — previous item must repeat 0 or 1 times
- `{n}` — previous item must appear n times

EXAMPLE PROBLEM

Replace any price in whole dollars (written like `$2` or `$1999`) with the string `-PRICE-`.

Note: `$` is a special character. To match a dollar sign, use `\$`.

MATCHING AND SEARCHING

What if you don't want to replace a regex, just find it?

- `re.match(pattern,string)` — does `string` begin with a match to `pattern`? Return a match object or `None`.
- `re.search(pattern,string)` — does `string` contain a match to the `pattern`? Return a match object or `None`.
- `re.findall(pattern,string)` — return a list of all non-overlapping matches *as strings*.

MATCH OBJECTS

If a match is found, then the match object has a method `.group()` that returns the full text of the match.

`.start()` and `.end()` return the indices where the match begins and ends in the string.

PARENTHESES

A part of a pattern in parentheses is a **group**. A group is treated as a unit for operators like `+`, `*`, `?`.

e.g. pattern `(ha)+` matches `ha` or `haha` or `hahaha` but does not match `Haha` or `h` or `hah`.

Matched groups are available from the match object using `.group(1)`, `.group(2)`, etc..

EXAMPLE PROBLEM

Find all of the phone numbers in a string that are written in the format `319-555-1012`, and split each one into area code (e.g. `319`), exchange (e.g. `555`), and line number (e.g. `1012`).

REFERENCES

- In *Downey*:
  - Regular expressions are not discussed.
- Google's free online Python course has a unit on regular expressions.
  - This course was developed for Python 2, so calls to `print` are lacking parentheses. Otherwise, the code should work.
- The documentation of the `re` module is good as a reference, not ideal to learn from.

REVISION HISTORY

- 2020-10-29 Move unused slides to Lecture 29
- 2020-10-27 Initial publication