Regular Expressions

Python II

Why

- A frequent programming task is “filtering”
  - Retain only those records that fit a certain pattern
  - Typical part of big data and analytics applications
  - Example for text processing

Why

- Whenever you deal with text processing
  - Think about whether you want to use regular expressions

Why

- Regular Expressions are a theoretical concept that is well understood
- Many programming languages have a module for regular expressions
    - Usually, very similar syntax and semantics
- We can use ad hoc solutions, but regular expressions are almost always faster

How

- Usually, we want to compile a regular expression
  - This allows for faster scanning
    - Compilation cost time
      - But usually amortized very quickly
- Python regular expressions are in module re
  - Use `p=re.compile('?')`
    - Where the question mark is the search string

How

- A Python regular expression is a string that defines the search
- The string is compiled
- After compilation, a match, search, or findall is performed on all strings
    - The output is None if the regular expression is not matched
    - Otherwise, depending on the function, it provides the parts of the string that match

A first example

- In a regular expression, most characters match themselves
  - Unless they are “meta-characters” such as *, \, ^
- E.G.: Find all lines in “alice.txt” with a double hyphen
- Regular expression is `'--'`
- Read in all lines of the text file, find the ones that match
  - Need to use search, because match only matches at the beginning of a string

A first example

```
import re

p = re.compile('--')

def match1():
    with open("alice.txt") as infile:
        line_count = 0
    for line in infile:
        line_count+=1
        line = line.strip()
        if p.search(line):
            print(line_count, line)
```

- Import re
- Compile the regular expression
- Match lines with .search()

Using raw strings

- A raw string is a string preceeded with a letter r:
  - `print(r'Hello World')`
- The difference to a normal string is that the escape character always means the escape character itself.
  - `print(r'\tHello')` prints out `\tHello`
  - `print('\tHello')` prints out `Hello` after a tab.
- This can be very useful because we might on occasion have to escape the escape character several times.

Matching

- Characters are the easiest to match
  - Find all words in lawler.txt (a large list of English words) with a double “oo”
    - Just change the expression
    -
```
import re

p = re.compile('oo')

def match1():
    with open("lawler.txt") as infile:
        line_count = 0
        for line in infile:
            line_count+=1
            line = line.strip()
            if p.search(line):
                print(line_count, line)
```

Matching

- Letters and numbers match themselves
- But are case sensitive
- Punctuation marks often mean something else.

Matching

- Square brackets [ ] mean that any of the enclosed characters will do
  - Example: [ab] means either 'a' or 'b'
- Square brackets can contain a range
  - Example: [0-5] means either 0, 1, 2, 3, 4, or 5
- A caret ^ means negation
  - Example: [^a-d] means neither 'a', 'b', 'c', nor 'd'

Self Test

- Find all lines in a file that have a double 'e'

Self Test Solution

```
import re

p = re.compile(r'ee')

def match_ee(filename):
    with open(filename) as infile:
        for line in infile:
            if p.search(line):
                print(line.strip())
```

Self Test 2

- Find all lines in a file that have a double-'ee' followed by a letter between 'l' (el) and 'n'

Self Test 2 Solution

```
import re

p = re.compile(r'ee[l-m]')

def match_ee(filename):
    with open(filename) as infile:
        for line in infile:
            if p.search(line):
                print(line.strip())
```

The only difference is in the regular expressions where we have now a range of letters.

Matching: Wild Cards

- Wild Card Characters
  - The simplest wild card character is the period / dot: “.”
  - It matches any single character, but not a new line
  - Example: Find all English words using Lawler.txt that have a patterns of an “a” followed by another letter followed by “a”
  - Solution: Use `p = re.compile('a.a')`

Matching: Wild Cards

- If you want to use the literal dot `'.'` you need to escape it with a backslash
- Example: To match “temp.txt” you can use `'t...\.txt'`
  - This matches any file name that starts with a t, has three characters afterwards, then a period, and then txt.

Matching: Repetitions

- The asterisks repeats the previous character zero or more times
  - Example: `'\.[a-z]*'` looks for a period, followed by any number of small letters, but will also match the simple string `'.'`
- The plus sign repeats the previous character one or more times.
  - Example: `'uni[a-z]+y'` matches a string that starts with 'uni' followed by at least one small letter and terminating with 'y'
    - This is difficult to read, as the + looks like an operation

Matching: Repetitions

- Braces (curly brackets) can be used to specify the exact number of repetitions
  - 'a{1:4}' means one, two, three, or four letters 'a'
  - 'a{4:4}' means exactly four letters 'a'

Self Test

- Write a regular expression that matches the name of any Python file.
    - Notice that ".py" is not a valid Python file. There must be something before the dot.

Self Test Solution

```
p2 = re.compile(r'.+\.py')
def match2():
    with open("temp.txt") as infile:
        line_count = 0
    for line in infile:
        line = line.strip()
        if p2.search(line):
            print(line)
```

Matching

- \w stands for any letter (small or capital) or any digit
- \W stands for anything that is **not** a letter or a digit
- Example: Matching “n”+non-letter/digit+”t”

```
"Speak English!" said the Eaglet. "I don't know the meaning of half
They were indeed a queer-looking party that assembled on the bank
```

- `p = re.compile('n\\Wt')`
  - We need to double escape the backslash using normal Python strings
- `p = re.compile(r'n\Wt')`
  - Or use a “raw string” (with an “r” before the string)
  - In a raw string, the backslash is always a backslash

Matching

- \s means a white space, newline, tab
- \S means anything but a white space, newline, or tab
- \d matches a digit
- \t matches a tab
- \r matches a return

Regular Expression Functions

- Once compiled a regular expression can be used with
  - **`match()`** matches at the beginning of the string and returns a match object or None
  - **`search()`** matches anywhere in the string and returns a match object or None
  - **`findall()`** matches anywhere in the string and **does not return a match object**

Match Objects

- A match object has its own set of methods
  - `group()` returns the string matched by the regular expression
  - `start()` returns the starting position of the matched string
  - `end()` returns the ending position
  - `span()` returns a tuple containing the (start, end) positions of a match

Regular Expression Gotcha

- Regular expression matching is **greedy**
  - Prefers to match as much of the string as it possibly can
- Example:

```
p3 = re.compile(r'.+\.py')
print( p3.search("This file, hello.py and this file
world.py are python files"))
```

  - Prints out

```
<re.Match object; span=(0, 42), match='This file,
hello.py and this file world.py'>
```

Non-Greedy Matching

- We can use the question mark qualifier to obtain a non-greedy match.
  - `p = re.compile('o.+?o')`
- Finds all non-overlapping, minimal instances

Advanced Topics

- In this module we only scratched the surface.
- There is excellent online documentation if you need more information
- But this should be sufficient to do simple tasks such as data cleaning and web scraping