Lecture #13: Regular Expressions

Facebook... *sigh*

On Languages...

- **Human (Natural Language)**
  - Developed by (social) evolution
  - Used to communicate between humans
  - Change “brain state” of recipient of communication

- **Mathematics**
  - Formal language developed out of philosophy
  - Syntax (structure) and semantics (meaning) well defined
  - Used to communicate scientific results between humans more rigorously

- **Programming Languages**
  - Formalized grammar allows automatic translation
  - Syntax and semantics unambiguous but limited
  - Used to communicate between humans and computer

Computational Concepts Toolbox

- Data type: values, literals, operations
- Expressions, Call expression
- Variables
- Assignment Statement
- Sequences: tuple, list
- Dictionaries
- Data structures
- Tuple assignment
- Function Definition Statement
- Conditional Statement
- Iteration: list comp, for, while
- Lambda function expr.

Higher Order Functions
  - as Values, Args, Results
- Higher order function patterns
  - Map, Filter, Reduce
  - Function factories
- Recursion
  - Linear, Tail, Tree
- Abstract Data Types
- Mutation
- Iterators and Generators
- Object Oriented Programming, Classes
- Exceptions
- Declarative Programming
  - Regular Expressions

Speaking of Facebook...

https://www.youtube.com/watch?v=bqWuioPHhz0

http://www.teachingprivacy.org
Syntax, Grammar, Semantics

• Syntax (programming language):
  – Set of rules that defines the combinations of symbols that are considered to be a correctly structured document or fragment in a programming language.

• Grammar:
  – Formalism (language) that defines the syntax of a programming language.

• Semantics:
  – Consequence (meaning) attached to a sequence of symbols.

Grammars (Chomsky)

• Grammars consist of:
  – Terminals (literals)
  – Non-terminals
  – Production rules

• Only define Syntax!

Example:

Another Grammar

\[ S \rightarrow aSb \]
\[ S \rightarrow \epsilon \]

- S non-terminal
- a,b terminal
- Epsilon: empty!

Question: What are valid words in this grammar?

Recap: Language Structures (Python)

• Variables and literals
  – with some internal representation, e.g. Integers, Floats, Booleans, Strings, …
  – In Python: Implicit data types!

• Operations on variable and literals of a type
  – e.g. +, *, -, /, %, //, **
  – ==, <, >, <=, >=

• Expressions are valid well-defined sets of operations on variables and literals that produce a value of a type.
  – x=4*3
Recap: while statement – iteration control

- Repeat a block of statements until a predicate expression is satisfied

```plaintext
<initialization statements>
while <predicate expression>:
    <body statements>
<rest of the program>
```

Grammar Hierarchy

- Programming Languages usually context-free
- Many filter tools for data usually regular languages (i.e. form regular expressions)

<table>
<thead>
<tr>
<th>Language</th>
<th>Grammar</th>
<th>Automaton</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>A → a</td>
<td>Finite state machine</td>
</tr>
<tr>
<td></td>
<td>A → aB</td>
<td></td>
</tr>
<tr>
<td>Context-free</td>
<td>A → αβ</td>
<td>Non-deterministic pushdown automaton</td>
</tr>
<tr>
<td>Context-sensitive</td>
<td>αAβ → αγβ</td>
<td>Linear-bounded non-deterministic Turing machine</td>
</tr>
<tr>
<td>Recursively enumerated</td>
<td>α → β</td>
<td>Turing machine</td>
</tr>
<tr>
<td></td>
<td>no restrictions</td>
<td></td>
</tr>
</tbody>
</table>

Grammar Hierarchy

- Easy to parse
- Semantics typically:
  - Find <string>
  - Find <string1>/Replace with <string2>
- Widely available in:
  - Python
  - Unix command line (bash, flex, sed)
  - Various editors (emacs, vi)
  - Most data science tools

- Important mechanism for ‘cleaning’ or ‘extracting’ data
Regular Expressions: Math

• Invited by S. Kleene
• Most important operator: * (say ‘star’ or Kleene star)
• s* means: ‘s’ n-times (for varying n)

Regular Expressions: Python

• re package
• Regular expressions are compiled into object
• Then various methods are available:

```
>>> import re
>>> p = re.compile('s*')
>>> p
re.compile('s*')
>>> print(p.match(''))
None
>>> m = p.match('sssss')
>>> m
<_sre.SRE_Match object; span=(0, 5),
match='sssss'>
```

More on REs in Python

```
>>> re.split('W+', 'Words, words, words.')
['Words', 'words', 'words']
>>> re.split('W+', 'Words, words, words.')
['Words', 'words', 'words', 'words', 'words']
>>> re.split('[a-f]*', '0a3B9', flags=re.IGNORECASE)
['0', '3', '9']
```

Full RE syntax

```
Summary of Regular Expression Patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atoms</td>
<td>Match any single character</td>
</tr>
<tr>
<td>Quantifiers</td>
<td>Match any character that appears any</td>
</tr>
<tr>
<td></td>
<td>number of times</td>
</tr>
<tr>
<td>Grouping operator</td>
<td>Match enclosed text</td>
</tr>
<tr>
<td>Backreference</td>
<td>Match previous backreference segment</td>
</tr>
<tr>
<td>Character class</td>
<td>Match character in specified range</td>
</tr>
<tr>
<td>Digit character class</td>
<td>Match digit (0-9)</td>
</tr>
<tr>
<td>Non-digit character class</td>
<td>Match non-digit</td>
</tr>
<tr>
<td>Alphanumeric character class</td>
<td>Match alphanumeric (0-9, a-f)</td>
</tr>
<tr>
<td>Whitespace char class</td>
<td>Match whitespace (space, tab)</td>
</tr>
<tr>
<td>Wildcard character</td>
<td>Match any character</td>
</tr>
<tr>
<td>Beginning of line</td>
<td>Match beginning of line</td>
</tr>
<tr>
<td>End of line</td>
<td>Match end of line</td>
</tr>
<tr>
<td>Word boundary</td>
<td>Match end of word</td>
</tr>
<tr>
<td>Alternative operator</td>
<td>Match any alternative</td>
</tr>
</tbody>
</table>

Constants

- re.IGNORECASE
- re.DOTALL
- re.UNICODE
- reVERBOSE
```

04/13/18 UCB CS88 Sp18 L13
Summary: RegEx

• Programming Languages are very formally defined using grammars
• Types of grammars need different complexity
• Regular expressions very simple and effective tool for data filtering
• More in the lab!

• Next Lecture: Information and Bits