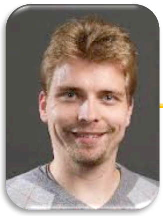


Computational Structures in Data Science

UC Berkeley EECS
Adj. Assistant Prof.
Dr. Gerald Friedland

Lecture #13: Regular Expressions

Facebook... *sigh*

Speaking of Facebook...

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

http://www.teachingprivacy.org

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

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

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:**

terminals
{generate, hate, great, green, ideas, linguists}
nonterminals
{*SENTENCE, NOUNPHRASE, VERBPHRASE, NOUN, VERB, ADJ*}
production rules
SENTENCE → *NOUNPHRASE VERBPHRASE*
NOUNPHRASE → *ADJ NOUNPHRASE*
NOUNPHRASE → *NOUN*
VERBPHRASE → *VERB NOUNPHRASE*
VERBPHRASE → *VERB*
NOUN → ideas
NOUN → linguists
VERB → generate
VERB → hate
ADJ → great
ADJ → green

Another Grammar

$$S \rightarrow aSb$$
$$S \rightarrow \varepsilon$$

- **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`

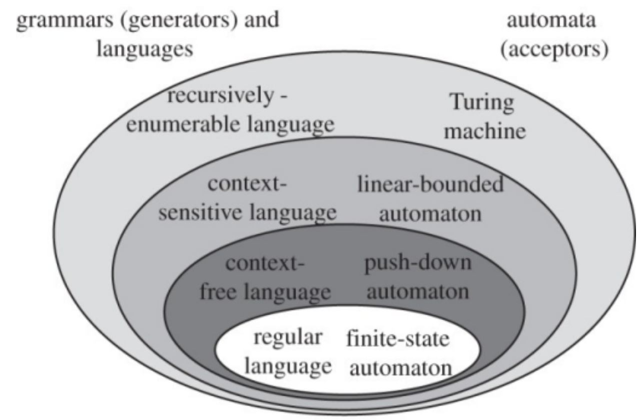
Recap: `while` statement – iteration control

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

```
<initialization statements>
while <predicate expression>:
   <body statements>

<rest of the program>
```


Grammar Hierarchy



the traditional Chomsky hierarchy

Grammar Hierarchy

Language	Grammar	Automaton
Regular	$A \to a$ $A \to aB$	Finite state machine
Context-free	$A \to \gamma$	Non-deterministic pushdown automaton
Context-sensitive	$\alpha A \beta \to \alpha \gamma \beta$	Linear-bounded non-deterministic Turing machine
Recursively enumerable	$\alpha \to \beta$ no restrictions	Turing machine

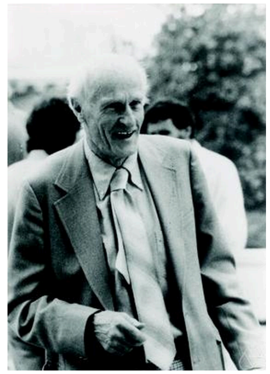
- Programming Languages usually context-free
- Many filter tools for data usually regular languages (i.e. form regular expressions)

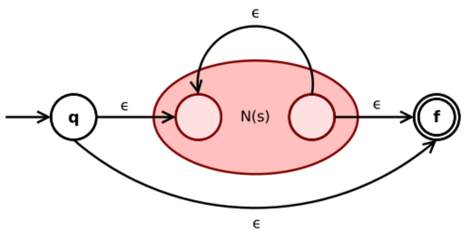
Regular Expressions

- 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', '.', '']
>>> re.split('\W+', 'Words, words, words.', 1)
['Words', 'words, words.']
>>> re.split('[a-f]+', '0a3B9', flags=re.IGNORECASE)
['0', '3', '9']

>>> re.split(’(\W+)’, ’...words, words...’)
['', ’...’, ’words’, ’, ’, ’words’, ’...’, '']

>>> re.split(’x*’, ’foo’)
[’foo’]
>>> re.split(”(?m)^$”, ”foo\n\nbar\n”)
[’foo\n\nbar\n’]
```


Full RE syntax

Summary of Regular Expression Patterns

Atoms

- Plain symbol: ...
- Escape: \
- Grouping operators: ( )
- Backreference: \#, \##
- Character class: [ ]
- Digit character class: \d
- Non-digit character class: \D
- Alphanumeric char class: \w
- Non-alphanum char class: \W
- Whitespace char class: \s
- Non-whitespace char class: \S
- Wildcard character: .
- Beginning of line: ^
- Beginning of string: \A
- End of line: $
- End of string: \Z
- Word boundary: \b
- Non-word boundary: \B
- Alternation operator: |

Constants

- re.IGNORECASE re.I
- re.LOCALE re.L
- re.MULTILINE re.M
- re.DOTALL re.S
- re.UNICODE re.U
- re.VERBOSE re.X

Quantifiers

- Universal quantifier: *
- Non-greedy universal quantifier: *?
- Existential quantifier: +
- Non-greedy existential quantifier: +?
- Potentiality quantifier: ?
- Non-greedy potentiality quantifier: ??
- Exact numeric quantifier: {*num*}
- Lower-bound quantifier: {*min*,}
- Bounded numeric quantifier: {*min*,*max*}
- Non-greedy bounded quantifier: {*min*,*max*}?

Group-Like Patterns

- Pattern modifiers: (?Limsux)
- Comments: (?#...)
- Non-backreferenced atom: (?:...)
- Positive Lookahead assertion: (?=...)
- Negative Lookahead assertion: (?!...)
- Positive Lookbehind assertion: (?<=...)
- Negative Lookbehind assertion: (?<!...)
- Named group identifier: (?P<*name*>)
- Named group backreference: (?P=*name*)

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