Regular Expressions (Regex)

CSCI 1030U - Intro to Computer Science
@IntroCS

Randy J. Fortier
@randy_fortier

OntarioTech
UNIVERSITY

Outline

- Regular expressions (regex)
    - Regular languages
    - Kleene’s Theorem
    - Regex in Python

Regular Expressions

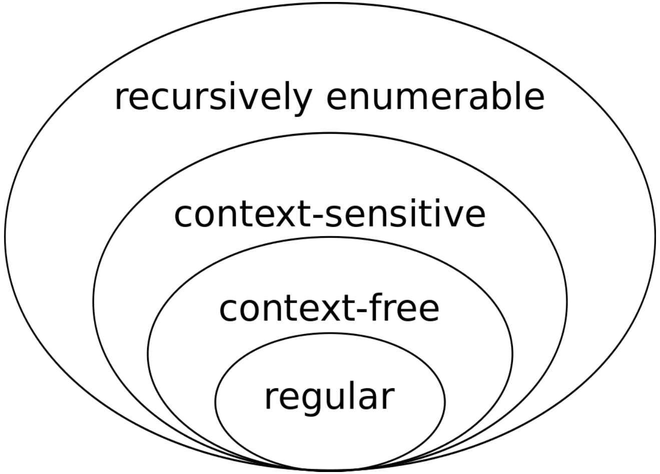
Regular Expressions

Regular Languages

- Regular languages are languages which can be described using regular expressions
  - Examples of regular languages:
    - E-Mail addresses
    - Phone numbers
    - Variable names
    - Comma-separated values

Other Languages

- Other languages, e.g. context-free languages, are more expressive than regular languages
  - Examples of context-free languages:
    - XML, HTML
  - Examples of context-sensitive languages
    - Python (and most programming languages)
    - English (and all natural languages)



Chomsky's Hierarchy

Regular Expressions (Regex)

- Regular expressions are expressions consisting of very simple rules, which can efficiently recognize regular languages
  - Basic symbols:
    - `.` - match any single character
    - `|` - union (`or`)
    - `*` - closure (0 or more)
    - `()` - used to group sub-expressions
  - Example: `aa*|.b*`
    - Either a sequence of 1 or more `a`s, or any single character, followed by 0 or more `b`s

Regex Extensions

- A number of extensions have been added to regular expressions
  - These do not modify the expressibility of the expressions, but do make them more convenient/shorter
  - `+` - 1 or more
  - `?` - 0 or 1 (optional)
  - `{5}` - exactly five occurrences
  - `{2,5}` - between two and five occurrences
  - `[abc]` - matches any one of these characters
  - `[a-zA-Z]` - matches any one of these characters
  - `[^a-z]` - matches any character except one of these characters
  - `\s` - matches any whitespace character (space, tab, newline)
  - `\w` - matches any word character, identical to `[a-zA-Z0-9 ]`
  - `\d` - matches any digit character, identical to `[0-9]`

Regex Extensions

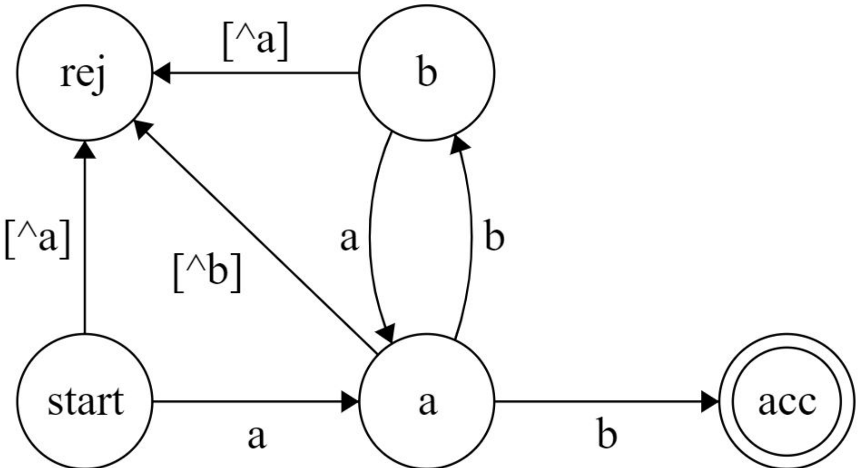
- Example: `[a-zA-Z_]\w*`
  - A letter or an underscore, followed by 0 or more letters, digits, or underscores (`x, average_mark`)
  - A variable name, a function name
  - e.g. `markSum1, class_average`
- Example: `(1-)?[0-9]{3}-[0-9]{3}-[0-9]{4}`
  - A north american phone number (with area code), optionally including a `1-` prefix
  - e.g. `1-905-721-8668`

Kleene's Theorem

- A deterministic finite state automaton has the same expressibility as a regular expression (extensions or not)
- In more formal language:
  - A language, L, is regular iff it can be recognized by a deterministic finite state automaton

Regular Expressions as FSAs

- With Kleene’s theorem in mind, we can revisit regular expressions
  - Example: `a(ba)*b`



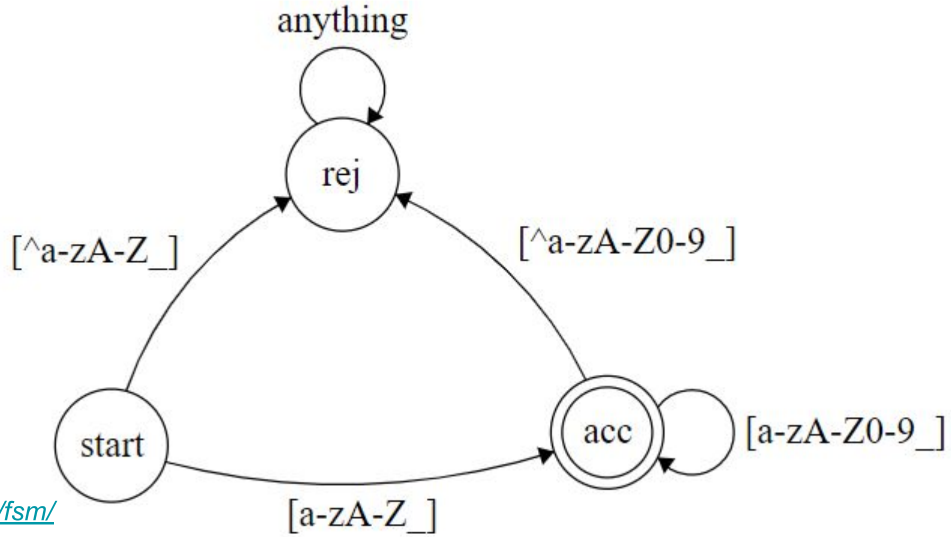
Regular Expressions as FSAs

- In addition to state transition diagrams, we can also represent FSAs using state transition tables:
  - Example: `a(ba)*b`

Input	Current State	New State
anything but 'a' `[^a]`	start	rej
'a'	start	a
'b'	a	b
anything but 'b' `[^b]`	a	rej
'a'	b	a
anything but 'a' `[^a]`	b	rej

Regular Expressions as FSAs

- With Kleene's theorem in mind, we can revisit regular expressions
  - Example: `[a-zA-Z_]\w*`   `(\w -> [a-zA-Z0-9_])`

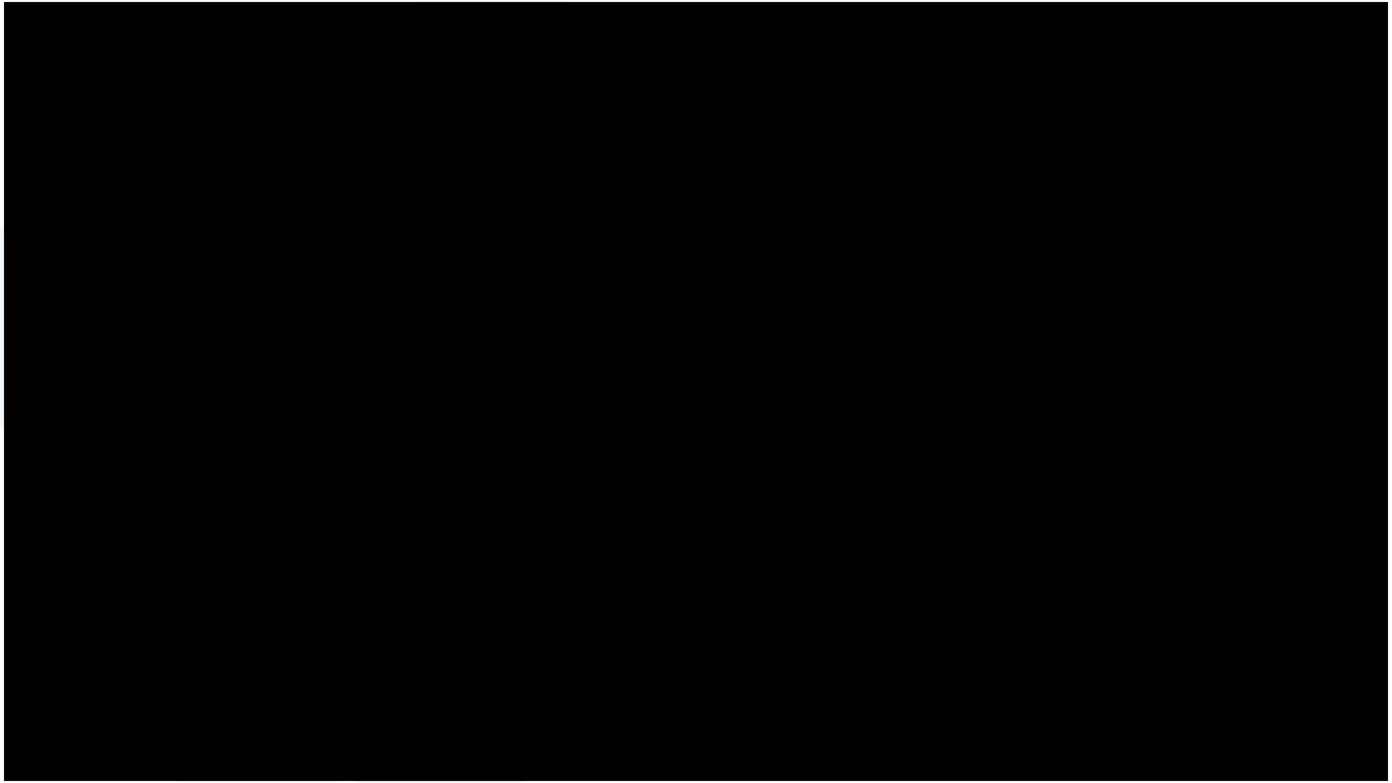


Drawn using http://madebyevan.com/fsm/

Animation - FSA for a Variable

Variable FSA - Video Example

Variable FSA - Video Example



Regular Expressions in Python

OntarioTech UNIVERSITY

Regex in Python

- The `re` package in Python allows you to easily match strings against regular expressions
  - `regex.match(string)` - Find matches at the start of `string`
  - `regex.search(string)` - Find matches within `string`
  - `regex.findall(string)` - Find all matches within `string`, returned as list
  - `regex.finditer(string)` - Find all matches within `string`, returned as an iterator (can be used in for loops)
  - `re.sub(regex, subst, string)` - Substitutes all matches of `regex` within `string`, with `subst`, returned as a string
  - `re.split(regex, string)` - Splits up `string`, using anything matching `regex` as a separator, returned as a list of strings between the separators

Regex in Python

- Example:

```
import re
nameRE = re.compile('[A-Z][a-z]*')
match = nameRE.match('Benjamin Button')
if match:
    print('Start:     ', match.start())
    print('End:       ', match.end())
    print('Text:      ', match.group())
print('All names: ', nameRE.findall('John Jonah Jameson'))
```

Regex in Python

- Example:

```
import re
phoneRE = re.compile('^(1-)?[0-9]{3}-[0-9]{3}-[0-9]{4}$')
match = phoneRE.match('905-721-8668')
if match:
    print('Start:     ', match.start())
    print('End:       ', match.end())
    print('Text:      ', match.group())
```

https://docs.python.org/3.8/howto/regex.html

Regex in Python

- Example:

```
import re
phoneRE = re.compile('^(1-)?[0-9]{3}-[0-9]{3}-[0-9]{4}$')
match = phoneRE.search('My phone number is 905-721-8668.')
if match:
    print('Start:     ', match.start())
    print('End:       ', match.end())
    print('Text:      ', match.group())
```

https://docs.python.org/3.8/howto/regex.html

Coding Exercise 07b.1

- Write a Python program to recognize a binary number of 8 or 16 bits

Coding Challenge 07b.1

- Write a Python program to recognize a simple E-Mail address
  - e.g. bsmith@gmail.com

- Not challenging enough for you?
  - Modify your regular expression to include more complicated E-Mail addresses:
    - e.g. randy.fortier@ontariotechu.net
    - e.g. candy_canes1@sweets.co.uk

OntarioTech UNIVERSITY

Wrap-up

- Regular expressions (regex)
  - Regular languages
  - Kleene’s Theorem
  - Regex in Python

Coming Up

- File I/O
  - Reading from files
  - Writing to files
- Exceptions
  - Catching exceptions
  - Throwing exceptions
  - Custom exceptions