UNIVERSITY of VIRGINIA

CS 1112: Introduction To Programming

Regular Expressions (RegEx)

Dr. Nada Basit // basit[*at*]Virginia[*dot*]edu

Friendly Reminders

- Your ***safety*** and ***comfort*** is important!
  - If you choose to wear a mask you are welcome to do so
  - *We will interpret wearing a mask as being **considerate and caring** of others in the classroom (not that you are sick), and realize that some may choose to mask to **remain distanced***

- Remember to always be **kind**, **respectful**, **supportive**, **compassionate** and **mindful of others**! ☺

- Be an ***active*** participant in your learning!
  You're welcome and ***encouraged*** to ask questions during class!

- If you feel ***unwell***, or think you are, please stay home
  - *Contact us!* *We will work with you!*
  - Get some rest ☺
  - View the recorded lectures – *please allow 24-48 hours to post*

Announcements

- **Quiz 7** will be due by 11:00pm on Monday (***3/31/2025***)
- **PA06** is due by 11:00pm on **Thursday, April 3** (***Note the *different* day!***)
  - Giving you an extra day!
  - Grace period is Friday, April 4 (must submit prior to **11pm** on Friday!)
  - TA office hours are made available on Thursday and Friday **that week** to assist

Coming up…

- **Exam 2**: Monday, April 7, 2025 *(SDAC accommodations? Book time slot on 4/7!)*
  - *In-class*
  - *Closed-book/closed-notes/closed-PyCharm/closed-Internet/closed-Computer/closed-everything!*
  - *Duration: 1 hour and 15 minutes*

Regular Expressions

CS 1112 - Introduction to Programming

Suggestion: Take Notes!

While I always encourage you to take notes in this class

This class would be particularly useful if you took notes

We will be talking about various symbols and unique details that are important to remember, so it would be useful to take notes

As always: don't hesitate to raise your hand, stop me, and ASK QUESTIONS! ☺

Regular Expressions (RegEx)

- Our knowledge base is expanding so rapidly. With so much data its hard to search and learn from this data. This has led to the rise of interdisciplinary cooperation
- Regular Expressions are an example of this cooperation
- An American mathematician **Stephen Kleene** (1909-1994) was the first person to introduce the term and the knowledge base in this subject
- After S. Kleene's initial work, the concept and application of Regular expressions has been greatly developed and enriched by Mathematicians, Language experts and computer scientists

Regular Expressions (RegEx)

- It should be no surprise that various rules and components of Regular expressions look like part of Algebra or computer code (algorithms)

- The basic ideas of regular expressions is quite simple. However, with advanced rules and regulations, RE have become very powerful tools being used by computer scientists, mathematicians and language experts

RegEx: Introduction

Here are 3 *ways* of defining a RegEx

- A regular expression (regex) is a way for a computer user or programmer to express how a computer program should look for a specified pattern in text. It’s a codified method of searching in a file
- In theoretical computer science and formal language theory, a regular expression is a sequence of characters that forms a search pattern
- A regular expression represents a pattern-matching rule for identifying content in a file

RegEx: Introduction

- Without using the fancy term RegEx, we all have used regular expressions since we bought our first computer. Almost all of us have used the characters or strings like “?” , “*”, “.doc”
- For example, a search request for “M?ry” will display all the occurrences of Mary or Mery. A search string “Miche*” will display all strings beginning with “Miche” including Michel, Michell and Michelle.
- In a Windows environment, the search key “*.docx” will display all the files with their names ending in “.docx”

RegEx: Introduction

- The Science of RegEx is the enhancement of these simple concepts
- With Algebra like rules and complex designs of search strings, RegEx have become very powerful and found their way in diverse applications
- A typical search and replace operation requires the *exact* text that matches the intended search result. This techniques does not have the flexibility of regular expressions

RegEx: Introduction

Some of the common applications of RegEx are:

- Test an input string
- Search, delete, replace text
- Intelligently recognize the contents of some given text: Social security number, telephone number, address etc
- Useful in some computer languages
- Validating email and/or password formats
- Lexical analysis of text for various applications including compilers

RegEx: The Simplest forms

- The most basic and simplest form of RegEx consist of
  - A single literal character
  - A string (collection) of alphanumeric characters

- The RegEx “`A`” will match 1st character in **A**pple
- The RegEx “`a`” will match 2nd character in c**a**t
- The RegEx “`cat`” matches cat in *Her cat is ginger*
- The RegEx “`MATH101` will match MATH101 in *MATH101 is interesting*
- The RegEx “`Cat`” will not match cat in “*Her cat is ginger*”- **Why?**

RegEx: Simple and advanced

- Literal characters and strings:
- We all have used them
  - Simple and easy
  - *Limited search capability*

- Modern set of RegExs:
  - Not so simple
  - Formula like
  - Flexible & powerful

RegEx: Bracket Expression [ ]

- Bracket expression matches a single character that is contained within the brackets

- **Gr [ae] y** will match Gray or Grey
- **S [iou]n** will match Sin, Son or Sun

- Particularly useful for searching a word that can be spelled in more than one way (e.g., Gray or Grey)!

Bracket Expression [ ] and hyphen

- A hyphen can be used to denote a range of characters:

- [A-Z] will match “A”, “B”, …, “Z”
- [a-z] will match any lowercase letter
- [0-9] will match “0”, “1”, …, “9”
- [A-Za-z] will match both upper and lower case letters
- [A-Za-z0-9] will match upper & lower case letters and 0-9
- [abcx-z] will match “a”, “b”, “c”, “x”, “y”, “z”
- [ab7-9] will match “a”, “b”, “7”, “8”, “9”

RegEx: Metacharacters

- Bracket expression we just discussed is a matacharacter
- There are many other types of metacharacters. The most important are:
  - Dot .
  - Caret ^
  - Question mark ?
  - Plus symbol +
  - Asterisk symbol *
  - Vertical bar |

RegEx: use of Metacharacter . (dot)

- Matches any single character when **not** inside square brackets:

- **X.Z** will match XAZ, XaZ, XBZ, ZbZ, XYZ etc

- However [**a.c**] will match “a”, “.”, “c”
(will match any *one* of the 3 characters – the dot included)

RegEx: use of Metacharacter ^ (caret) inside []'s

- Som e things. C s hand
- Car ter th d with ce e [ ]'s

DID YOU SAY CARROT?

- [^abc] will match any character other
- [ab^cd] will match 'a', 'b', '^' (*charact*
- [^0-9] will match any non-numeric ch
- [^a-z] y character other t a to z
- N f [ ]s) means "t tring or atches any line or string *beginning* with 'x'

NO...

Image: pngtree

RegEx: use of Metacharacter ^ (caret) inside [ ]'s

- Sometimes we want to exclude things. Caret ^ comes handy
- Caret will match a single character that is **not** contained within the brackets - *has to be placed at the beginning of the [ ]'s*

- **[^abc]** will match any character other than "a", "b", "c"
- [ab^cd] will match 'a', 'b', '^' (***character***), 'c', and 'd'
- **[^0-9]** will match any non-numeric characters
- **[^a-z]** will match any character other than characters a to z
- Note: ^ alone (outside of [ ]s) means "the start of the string or line"   e.g. ^x matches any line or string *beginning* with 'x'

RegEx: use of Metacharacter ?

- Use of ? is different in RegEx than in windows
- Question mark matches the *preceding* element zero or one times

- **ab?c** matches the following:
  - “ac” here the ‘b’ was matched *zero* times
  - “abc” here the ‘b’ was matched *one* time
  - “abbc” would not match this

RegEx: use of Metacharacter + (plus)

- The plus sign matches the *preceding* element one or more times

- **ab+c** matches the following:
- "abc" matches 'b' one time
- "abbc" matches 'b' two times
- "abbbc" matches 'b' three times,
- … and so on
- Here, it will ***not*** match "ac" (min # of b's to be matched is 1)

RegEx: use of Metacharacter * (asterisk)

- The asterisk sign matches the *preceding* element or elements (pattern) zero or more times

- X(ab)* matches the following:

X	zero instances of "ab"
Xab	one instance of "ab"
Xabab	two instances of "ab"
Xababab	three instances of "ab"

- … and so on
- Here, it will ***not*** match "Xa" or "Xb" or "Xbbbbb" …

RegEx: use of Metacharacter |

- Use of vertical bar works quite example, many programming languages. It works as the logical OR statement

- For example, **abc|xyz** matches "abc" ***or*** "xyz"

- We can also have **multiple** vertical bars in a single statement such as: **a|b|x|y|z**
- It will match "a" *or* "b" *or* "x" *or* "y" *or* "z"

SUMMARY

- [] = brackets, matches a single character contained within
- [x-z] = hyphen, denotes a range of characters
- a.b = dot, matches any single character in that location
- [^abc] = caret, matches a single character that is not a, b, or c
- c? = preceding character (c) 0 or 1 times
- c+ = preceding character (c) 1 or MORE times
- c* = preceding character (c) 0 or MORE times
- a|b = vertical bar, means or