The Puzzling Quirks of Regular Expresssions

David Mertz, Ph.D.

2021-08-01

Contents

Preface

Regular expressions—sometimes given the playful back formation *regexen* or more neurally *regex*—are a powerful and compact way of describing patterns in text. Many tutorials and "cheat sheets" exist to understand their syntax and semantics in a formally correct manner. I encourage you to read some of those, if you have not already.

These brain teasers begin at a certain point where the formal descriptions leave off. As you work with regexen, you will find subtle pitfalls. A pattern that seems like it should obviously do one thing, actually matches something slightly different than you intended. Or perhaps a match pattern has "pathological" behavior and takes far too long. Or sometimes it is simply that a more concise pattern can also be clearer in describing what you actually wish to match.

A great many programming languages, libraries, and tools support regular expressions, with relatively minor variations in the syntax used. Such software includes `[efr]?grep sed`, `awk`, *Perl*, *Java*, *.NET*, *JavaScript*, *Julia*, *XML Schema*, or indeed, pretty much every other programming language via a library.

For this book, we will use Python to pose these puzzles. In particular, we will use the standard library module re. Often code samples are used in puzzles and in explanation; where I wish to show the output from code, the example emulates to the Python shell with lines starting with `>>>` (or continuing with `...`). Outputs are echoed without a prompt in this case. Where code defines a function that is not necessarily executed in the mention, only the plain code is shown.

While you are reading this book, I strongly encourage you to keep open an interactive Python environment. Many tools enable this, such as the Python REPL (read-evaluate-print-loop) itself, or the IPython enhanced REPL, or Jupyter notebooks, or the IDLE editor that comes with Python, or indeed most modern code editors and IDEs (integrated development environment). A number of online regular expression testers are also available, although those will not capture the details of the Python calling details. Explanations will follow each puzzle, but trying to work it out in code before reading it is worthwhile.

Quantifiers and Special Sub-Patterns

Solving the puzzles in this section will require you to have a good understanding of the different quantifiers that regular expressions provide, and to pay careful attention to when you should use sub-patterns (themselves likely quanitifed).

file=“scope”

file="scope2"

file=“endpoints”

file="config"

file="fasta"

Pitfalls and Sand in the Gears

As compact and expressive as regular expressions can be, there are times when they simply go disasterously wrong. Be careful to avoid, or at least to understand and identify, where such difficulties arise.

file=“catastrophic”

file=“dominoes”

file="dominoes2"

file=“linedraw”

Creating Functions using Regexen

Very often in Python, or in other programming languages, you will want to wrap a regular expression in a small function rather than repeat it inline.

file="count"

file="count2"

file=“naming”

file="poker1"

file=“poker2”

file="poker3"

file=“poker4”

file="poker5"

Easy, Difficult, and Impossible Tasks

Some things are difficult or impossible with regular expressions, and many are elegant and highly expressive. The puzzles in this section ask you to think about which situation each puzzle describes.

file="matchcount"

file="nodup"

file="ipv4"

file="number_sequence"

file="fibonacci"

file="primes"

file=“relprimes”