GENERAL ASSEMBLY

INTRO TO REGEX

Haley Boyan

Data Science Instructional Associate, General Assembly DC

LEARNING OBJECTIVES

- Understand what RegEx is used for
- List common syntax cues for RegEx
- Implement RegEx code in a Python script

WELCOME TO REGEX

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

WHAT IS REGEX?

WHAT IS REGEX?

- "RegEx" = Regular Expressions
- Specialized syntax for matching portions of text.
- Define a pattern, and search for parts of a string that match the pattern
- Effectively its own language
- Used in every major programming language

REGEX EXPRESSIONS

- A regular expression is a pattern of text that consists of:
    - Ordinary characters (for example, letters a through z)
    - Special characters, known as metacharacters

REGEX EXAMPLES

Regular Expression	Match	Do Not Match
“123”	“12345” “abc123def” “123”	“1a2b3c” “321”
“b"	“b” “abc” “bread”	“xyz” “c” “B”

REGEX SPECIAL CHARACTERS

- A period acts as a wild card, indicating that any single character (except a newline) can be put in its place

Regular Expression	Match	Do Not Match
“a.c”	“aac” “abc” “alchemy” “branch”	“add” “crash” “abbbbc”
“..t”	“bat” “habit” “oat”	“at” “it” “too”

REGEX SPECIAL CHARACTERS

- "^" to match the start of a string
- "$" to match the end of string.

Regular Expression	Match	Do Not Match
“^a"	“apple juice” “abc” “are you having fun”	“boat” “having a ball” “China”
“a$”	“China” “I’m going to China” “cba”	“abc” “cat” “literally anything where ‘a’ is not the last letter”

REGEX SPECIAL CHARACTERS

- “[ ]” (Square brackets) are used to indicate that any character within the square bracket can fill the space.

Regular Expression	Match	Do Not Match
“[bcr]”	bat cat rat	anything else

REGEX SPECIAL CHARACTERS

- "( )" Parentheses indicate a group of characters
- "(?=RE)" Positive lookahead (if the RE appears next in the string)
- "(?!RE)" Negative lookahead (if the RE does **not** appear next)

Regular Expression	Match	Do Not Match
"(abc)"	abc	anything else
"ab(?=c)"	abc cabc	ab c abd
"ab(?!c)"	ab c abd	abc cabc

REGEX SPECIAL CHARACTERS

- ‘*’: match 0 or more repetitions of the preceding RE, as many repetitions as are possible
- ‘+’: match 1 or more repetitions of the preceding RE
- ‘?’: match 0 or 1 repetitions of the preceding RE. ab? will match either ‘a’ or ‘ab’.
- {m}: **exactly** m copies of the previous RE should be matched
- {m,n}: match from m to n repetitions of the preceding RE, attempting to match as many repetitions as possible.
    - Omitting m specifies a lower bound of zero, and omitting n specifies an infinite upper bound.
    - The comma may not be omitted or the modifier would be confused with the previously described form.

Regular Expression	Match	Do Not Match
“ab*”	“a” “ab” “abbbbbb”	“ac” “bb” “ba”
“ab+"	“ab” “abb”	“ba” “b” “a”
“ab?”	“a” “ab”	anything else
“a{6}”	“aaaaaa"	anything else
“a{3,5}”	“aaa” “aaaa” “aaaaa”	
“a{4,}”	“aaaab” “aaaaaaaaaaaaaaaaab”	“ab” “aaaaa” “abaa”

GREEDINESS OF SPECIAL CHARACTERS

- The '*', '+', and '?' qualifiers are all greedy; they match as much text as possible
    - If the RE <.*> is matched against <a> b <c>, it will match the entire string, and not just <a>
- Adding ? after the qualifier makes it perform the match in non-greedy or minimal fashion; as few characters as possible will be matched
    - Using the RE <.*?> will match only <a>.

ESCAPING REGEX CHARACTERS

- Use "\" (backslash) to escape special characters
- Escaping a character indicates that the character should be treated as any other character would be

Regular Expression	Match	Do Not Match
“.”	“a” “cat” “strings with any character”	“”
“\.”	“www.google.com” “End of a sentence.”	“a” “cat” “anything without a period”

ESCAPING REGEX CHARACTERS

Notes:
- The capital letter version of these does the opposite of their given behavior
- These guidelines are for UTF encoding, and behave differently for 8-bit or ASCII encoding

- \number: Matches the contents of the group of the same number (starting from 1 to 99 - then it gets weird)
- \A: Matches only at the start of the string.
- \Z: Matches only at the end of the string.
- \b: Matches the empty string, but **only at** the beginning or end of a word (the end of a word is indicated by whitespace or a non-alphanumeric, non-underscore Unicode character)
- \d: Matches any Unicode decimal digit. This includes [0-9], and also many other digit characters.
- \s: Matches Unicode whitespace characters (which includes [\t\n\r\f\v], and also many other characters
- \w: Matches Unicode word characters; this includes most characters that can be part of a word in any language, as well as numbers and the underscore

Regular Expression	Match	Do Not Match
"(.+) "	"the the" "55 55"	"thethe"
"\bfoo\b"	"foo" "foo." "(foo)" "bar foo ba"	"foobar" "f003"
r'py\B'	python' 'py3'	py' 'py.' 'py!'

REGEX SPECIAL CHARACTERS

- To combine regular expressions, we use the "|" character

re.compile

- re.compile(pattern, flags=0): Compile a regular expression pattern into a regular expression object, which can be used for matching using its match() and search() methods
- The sequence
  - prog = re.compile(pattern)
  - result = prog.match(string)
- is equivalent to
  - result = re.match(pattern, string)
- using re.compile() and saving the resulting regular expression object for reuse is more efficient when the expression will be used several times in a single program.

REGEX COMMANDS

- re.search(pattern, string, flags=0)
    - Scan through string looking for the first location where the regular expression pattern produces a match, and return a corresponding match object. Return None if no position in the string matches the pattern.
- re.match(pattern, string, flags=0)
    - If zero or more characters at the beginning of string match the regular expression pattern, return a corresponding match object. Return None if the string does not match the pattern.
- If you want to locate a match anywhere in string, use search()

REGEX COMMANDS

- re.findall(pattern, string, flags=0)
    - Return all non-overlapping matches of pattern in string, as a list of strings. The string is scanned left-to-right, and matches are returned in the order found

- re.split(pattern, string, maxsplit=0, flags=0)
    - Split string by the occurrences of pattern

DEMO: USING REGEX

DEMO REGEX

- See Jupiter Notebook

GUIDED PRACTICE: BUTTERING UP REGEX

ACTIVITY: TITLE OF ACTIVITY

EXERCISE

DIRECTIONS

1. Create a variable called regex
2. Create a variable called “text,” and initialize it with:
   “Better not put too much butter in the batter, we wouldn’t want it to be bitter”
3. Create a regex that matches “Better” “butter” and “batter”, but not “bitter”

DELIVERABLE

Regex expression

ACTIVITY: TITLE OF ACTIVITY

EXERCISE

DIRECTIONS

1. Create a variable called regex
2. Create a variable called "text," and initialize it with:
   "Better not put too much butter in the batter, we wouldn't want it to be bitter"
3. Create a regex that matches "Better" "butter" and "batter", but not "bitter"

[Bb].tter(?=.)

DELIVERABLE

Regex expression

INDEPENDENT PRACTICE

REGEX AND REDDIT

ASKREDDIT DATA

- DEMO
- We'll be working with a dataset of the top 1000 posts on AskReddit in 2015

READING AND PRINTING THE DATASET

- Let's use the csv module to read and then print our dataset
- The csv module implements classes to read and write tabular data in CSV format.
- reader () returns a reader object which will iterate over lines in the given csvfile.

TESTING FOR MATCHES

- With re.search(regex, string), we can check if string is a match for regex
- If it is, it will return a match object. If it isn't, it will return None.

ACCOUNTING FOR INCONSISTENCIES

- In regular expressions, square brackets are used to indicate that any character within the square bracket can fill the space.
- For example, the regular expression "[bcr]at" would be matched by strings with the substrings "bat", "cat", and "rat", but nothing else.
- We can account for the inconsistency of people writing "of Reddit" versus "of reddit” using square brackets. []

YOU DO: 5 MINUTES

- In our dataset, some people tag serious posts as "[Serious]", and others as "[serious]". We should account for both capitalizations.
- Refine the code to count how many posts have either "[Serious]" or "[serious]" in their title.
- Assign the count to serious_count.

YOU DO: 5 MINUTES

- In our dataset, some users have tagged their post using "(Serious)" or "(serious)". We should account for both square brackets and parenthesis. We can do this using square bracket notation, and escaping "[", "]", "(", and ")" with the backslash.
- Refine the code to count how many posts are tagged as serious, using either square brackets or parenthesis.
- Assign the count to serious_count.

YOU DO: 10 MINUTES

- Use the "^" character to count how many posts have the serious tag at the beginning of their title. Assign this count to serious_start_count.
- Use the "$" character to count how many posts have the serious tag at the end of their title. Assign this count to serious_end_count.
- Use the "|" character to count how many posts have the serious tag at either the beginning or end of their title. Assign this count to serious_count_final.

TITLE

CREDITS

TITLE

RESOURCES/PRACTICE

- https://docs.python.org/3/library/re.html
- regexr.com
- https://regexcrossword.com/