REGULAR TEXT

Jed Rembold

Monday, November 4, 2024

ANNOUNCEMENTS

- HW8 posted!
  - Working with subqueries, dates and times
- Test 2 handed back hopefully on Wednesday
- Spending this week in class with text mining
- Polling: polling.jedrembold.prof

REVIEW QUESTION

Given the starting table called `rev`, what is the output of the query?

name	num
A	1
B	2
C	3
D	4
E	5

```
SELECT
  CASE
    WHEN num % 2 = 0 THEN name
    WHEN name > 'B' THEN 'D'
    ELSE 'A'
  END
FROM rev
WHERE num < 4
ORDER BY num DESC
LIMIT 1
```

TEXT POWER

- Time to focus on everything we can do with strings!
- Chapter topic fall into several main ideas:
  - Manipulating strings
  - More complicated pattern matching
  - Full text searching using normalization and lexemes
- All are geared around making using text and strings much more powerful and flexible

BASIC STRING OPERATIONS

STRINGY FUNCTIONS (CORE)

Function	Description
`str || str2`	Concatenates string 1 and string 2 together
`upper(str)`	Converts a string to all uppercase characters
`lower(str)`	Converts a string to all lowercase characters
`char_length(str)`	Returns the number of characters in the string
`position(str IN substr)`	Find the number of the character where the substring begins
`trim(opt chr FROM str)`	Removes the given characters from the string, optionally taking from the *leading* or *trailing* edge
`substring(str FROM n FOR l)`	Returns the portion of the string starting at position n and continuing for l characters

STRING FUNCTIONS (POSTGRES)

Function	Description
`initcap(`*str*`)`	Converts the first character of each word to uppercase, and the rest lower
`left(`*str*`,`*n*`)`	Returns the first n characters of the string
`right(`*str*`,`*n*`)`	Returns the last n characters of the string
`ltrim(`*str*`,`*chr*`)`	Remove the characters (space by default) from the start of the string
`rtrim(`*str*`,`*chr*`)`	Remove the characters (space by default) from the end of the string
`replace(`*str*`,`*from*`,`*to*`)`	Replaces all occurance of *from* in the string to *to*
`length(`*str*`)`	Returns the number of characters in the string
`substr(`*str*`, `*n*`, `*l*`)`	Returns the portion of the string starting at position n and continuing l characters

REGULAR EXPRESSIONS

ENHANCED PATTERN MATCHING

- We've already seen basic pattern matching with `LIKE` and `LIKE`
  - Some flexibility with wildcard characters: `%` and `_`
- To get (much) more flexibility, we need to pivot to something made for exactly this purpose: *regular expressions* (or *regex*)
- Regular expressions are a sequence of mostly single character symbols that denote exactly what patterns one could wish for
  - These sequences of characters can initially look very inscrutable! Stick with it!
- Regex's are useful all over, and supported in almost all programming languages as well. Learning at least the basics is time very well spent.

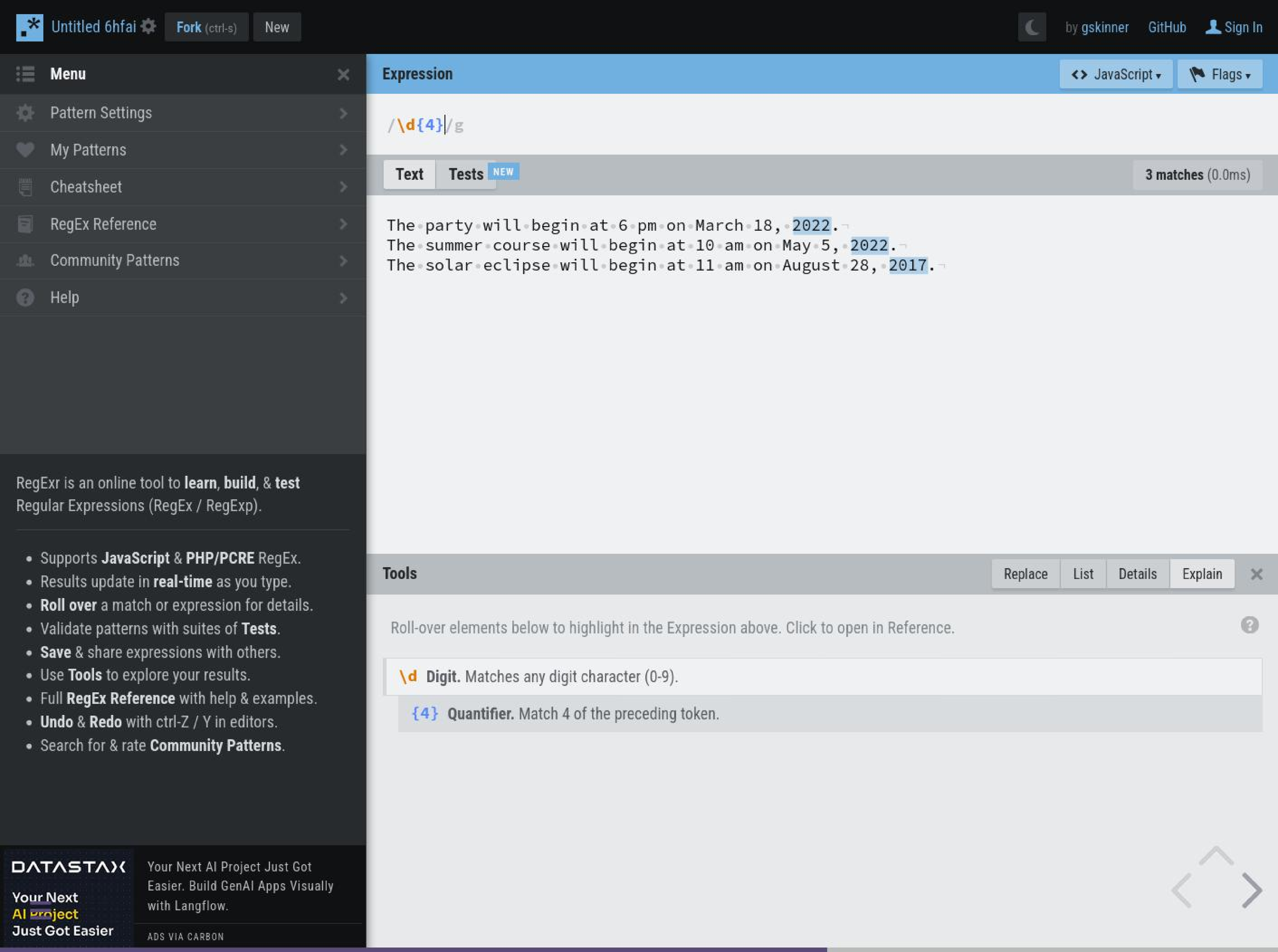
BASIC REGEX TERMS

Expression	Description
`.`	Matches *any* character except a new line (this can vary some in other implementations)
`[abc]`	Matches any character in the square brackets (a or b or c)
`[a-z]`	Matches a range of characters (all lowercase letters here)
`[^a-z]`	Caret negates what follows (so no lowercase letters here)
`\w`	Any word character, digit or underscore
`\d`	Any digit
`\s`	A space
`\t`	A tab character
`\n`	A newline character

Expression	Description
`^`	Match at the start of the string
`$`	Match at the end of the string
`?`	Get the preceding match 0 or one time
`*`	Get the preceding match zero or more times
`+`	Get the preceding match one or more times
`{m}`	Get the preceding match exactly m times
`{m,n}`	Get the preceding match between m and n times
`a | b`	Match on either a or b, where a and b are full matching expressions
`( )`	Create a capture group or set precedence
`(?: )`	Negate reporting a capture group

OTHER REGEX CONCEPTS

- If you ever want to match off a symbol that has special meaning in regex (a parenthese, for instance) you must *escape it* with a backslash: `\(`
- Reserved characters include: `{ } [ ] / \ + * . $ ^ | ?`
- Flags can be added at the end to tweak matching
  - `/i` means that matches will be case insensitive
  - `/g` means that all instances of the match will be returned, not just the first
  - `/m` allows the anchor characters (`^` and `$`) to operate on each line, not just across the entire string.



```
/\d{4}/g
```

```
The party will begin at 6 pm on March 18, 2022.
The summer course will begin at 10 am on May 5, 2022.
The solar eclipse will begin at 11 am on August 28, 2017.
```

Roll-over elements below to highlight in the Expression above. Click to open in Reference.

`\d` **Digit.** Matches any digit character (0-9).

`{4}` **Quantifier.** Match 4 of the preceding token.

RegExr is an online tool to **learn**, **build**, & **test** Regular Expressions (RegEx / RegExp).

- Supports **JavaScript** & **PHP/PCRE** RegEx.
- Results update in **real-time** as you type.
- **Roll over** a match or expression for details.
- Validate patterns with suites of **Tests**.
- **Save** & share expressions with others.
- Use **Tools** to explore your results.
- Full **RegEx Reference** with help & examples.
- **Undo** & **Redo** with ctrl-Z / Y in editors.
- Search for & rate **Community Patterns**.

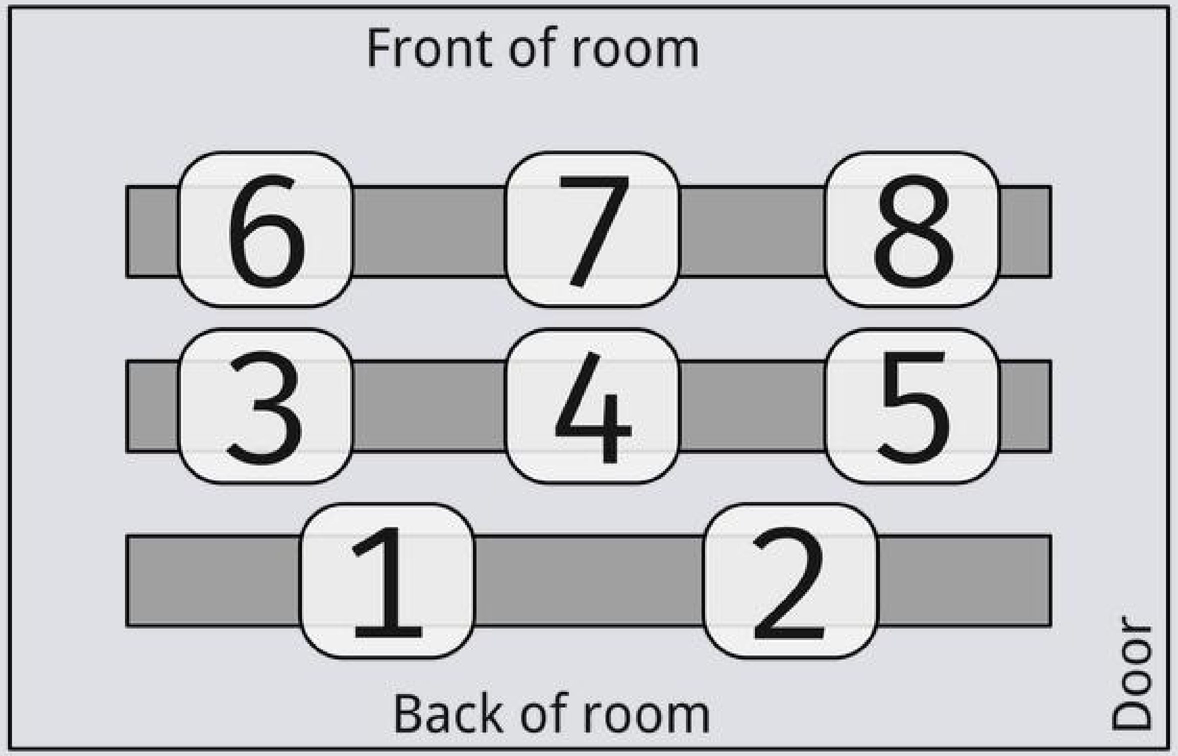
ACTIVITY

YOUR TURN!

- The link here has a nice sequence of short problems to test your skills against
- Most problems consist of:
  - Terms that you want to match correctly
  - Terms that you want to not match
  - Capture groups that you'd like to capture
- In the next slide groups, see how many you can figure out in the next 20 minutes

TODAY'S GROUPS

- Group 1: Evan, Harleen, Tippy
- Group 2: Dayton, Sam J, Michael
- Group 3: Greg, Jerrick, Mallory
- Group 4: Marcus, AJ, Matthew
- Group 5: Connor, Grace, Haley
- Group 6: Sergio, Tiffany
- Group 7: Aurora, Nick, Jordan
- Group 8: Hannah, Jack, Sam H



Group Areas

REGEX IN POSTGRES

BACK TO SQL

- One of the main ways we previously used pattern matching was for filtering
- You can also use regexes for pattern matching!
  - `~` is a case sensitive match using the following regex
  - `~*` is a case insensitive match using the following regex
  - Either can have a `!` in front to negate the search (where things do not match the regex)

```
SELECT colname
FROM tablename
WHERE colname ~ '[a-z]*\s\d{2}';
```

EXTRACTING DATA

- Another hugely common use of regex is to extract only the data you want from a much larger string
- This can be particularly useful when cleaning data or constructing useful database tables
- `regexp_match(str, regex)` returns the first matching instance in the string
  - What is returned is whatever is in any *capture groups* you may have included in your regex, or the entire match if there are no capture groups
  - Output is returned as an array, to allow for potentially multiple capture groups
  - If you just have one capture group and don't want it in an array, index it out using `[1]` at the end after wrapping entire expression in ()

```
SELECT (regexp_match('today is March 15, 2022', '\d{4}'))[1];
```