Exploring Regular Expression Usage and Context in Python

Carl Chapman, Kathryn T. Stolee*

Iowa State University, North Carolina State University

carlallenchapman@gmail.com, ktstolee@ncsu.edu

19 July, 2016

Why regular expressions?

- Regexes are everywhere! (we think...)

Why regular expressions?

- Regexes are everywhere! (we think...)
- Everyone writes regexes! (we think...)

Why regular expressions?

- Regexes are everywhere! (we think...)
- Everyone writes regexes! (we think...)
- Regexes are hard to read/write! (again, we think...)

Why regular expressions?

- So, we wanted to write a tool to support regex creation.

Why regular expressions?

- So, we wanted to write a tool to support regex creation.
- But...

Why regular expressions?

- So, we wanted to write a tool to support regex creation.
- But...

Regex feature usage references are missing!

Why regular expressions?

- So, we wanted to write a tool to support regex creation.
- But...

Regex feature usage references are missing!

- and...

Why regular expressions?

- So, we wanted to write a tool to support regex creation.
- But...

Regex feature usage references are missing!

- and...

We don't know how/when/why developers use regexes!

Research goals

1. RQ1: In what contexts do professional developers use regular expressions?
2. RQ2: How is the re module used in Python projects?
3. RQ3: Which regular expression language features are most commonly used in Python?
4. RQ4: How behaviorally similar are regexes across projects?

Research goals

1. RQ1: In what contexts do professional developers use regular expressions?
2. RQ2: How is the re module used in Python projects?
3. RQ3: Which regular expression language features are most commonly used in Python?
4. RQ4: How behaviorally similar are regexes across projects?

Regular expressions: The basics

- `(ab*c|yz*)$`
  - ✓ abbbbbbbc
  - ✓ y
  - ✓ abcy

Regular expressions: The basics

- `(ab*c|yz*)$`
  - ✓ abbbbbbbc
  - ✓ y
  - ✓ abcy
  - ✗ abcccc
  - ✗ yxw

Regular expressions: The basics

- `(ab*c|yz*)$`
  - ✓ abbbbbbbc
  - ✓ y
  - ✓ abcy
  - ✗ abcccc
  - ✗ yxw

- `(ab*c|yz*)`

Regular expressions: The basics

- `(ab*c|yz*)$`
  - ✓ abbbbbbbc
  - ✓ y
  - ✓ abcy
  - ✗ abcccc
  - ✗ yxw

- `(ab*c|yz*)`
  - ✓ abbbbbbbc
  - ✓ y
  - ✓ abcy
  - ✓ abcccc
  - ✓ yxw

Survey context

- 18 professional developers
- 9 years average development experience
- Small mobile payment management company
- 30 questions in a Google form

How frequently do developers use regexes?

- 50% – at least once per week
- Regexes are most frequently composed within command line and text editor tools
- 2 developers write more than 50 regexes in general programming languages (e.g., Java) annually
- Database queries using regexes were rare

Common regex activities

How often do you use regexes for...

Activity	Frequency
Locating content within a file or files	4.4
Capturing parts of strings	4.3
Parsing user input	4.0

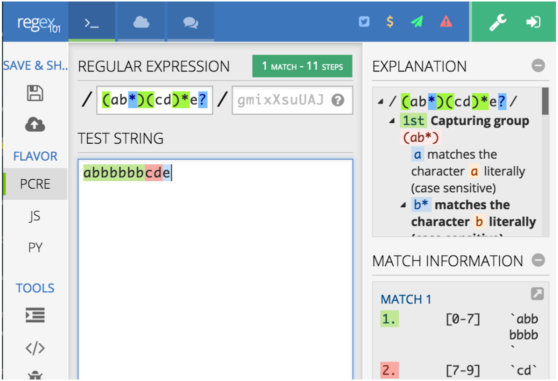
Key: 6 = very frequently, 5 = frequently, 4 = occasionally, 3 = rarely, 2 = very rarely, 1 = never

Testing regular expressions

Developers test regular expressions less often than other code.

Testing regular expressions

Developers test regular expressions less often than other code.



REGULAR EXPRESSION 1 MATCH - 11 STEPS

/ (ab*)(cd)*e? / gmixXsuUAJ

TEST STRING

abbbbbbcde

EXPLANATION

/ (ab*)(cd)*e? /
1st Capturing group (ab*)
a matches the character a literally (case sensitive)
b* matches the character b literally (case sensitive)

MATCH INFORMATION

MATCH 1

1.	[0-7]	`abbbbbb`
2.	[7-9]	`cd`

50% say they use testing tools like www.regex101.com

Pain points

hard to compose (11 = 61%)

...very difficult to write them since I‘ve never read up on them.

...trickiness to getting the expression right

Pain points

hard to compose (11 = 61%)

...very difficult to write them since I‘ve never read up on them.

...trickiness to getting the expression right

hard to read (7 = 39%)

long ones can be hard to read

Readability. Edge cases.

It is terrible to read (especially later after initial development)

Pain points

hard to compose (11 = 61%)

…very difficult to write them since I‘ve never read up on them.

…trickiness to getting the expression right

hard to read (7 = 39%)

long ones can be hard to read

Readability. Edge cases.

It is terrible to read (especially later after initial development)

inconsistency across implementations (3 = 17%)

Differences in implementation across languages

Some regexes work differently (or don‘t work) in some languages.

Notable Observations

- Regexes are composed fairly frequently by developers
- Testing regexes is less common than testing other code
- Developers find regexes hard to read and write

Notable Observations

- Regexes are composed fairly frequently by developers
- Testing regexes is less common than testing other code
- Developers find regexes hard to read and write

but....

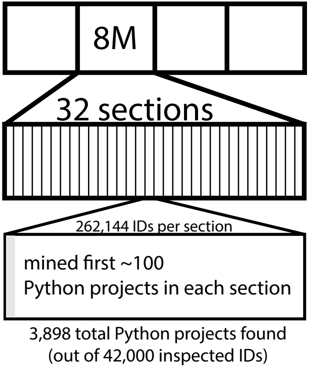
Notable Observations

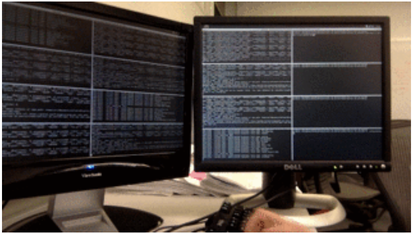
- Regexes are composed fairly frequently by developers
- Testing regexes is less common than testing other code
- Developers find regexes hard to read and write

but....

How do developers really use regexes?

Project selection with the GitHub API


8M
32 sections
262,144 IDs per section
mined first ~100
Python projects in each section
3,898 total Python projects found
(out of 42,000 inspected IDs)



Out of 3,898 pseudo-randomly selected Python projects, 1,645 (42%) contained one or more regex utilization.

In Python: Utilizations of the re module

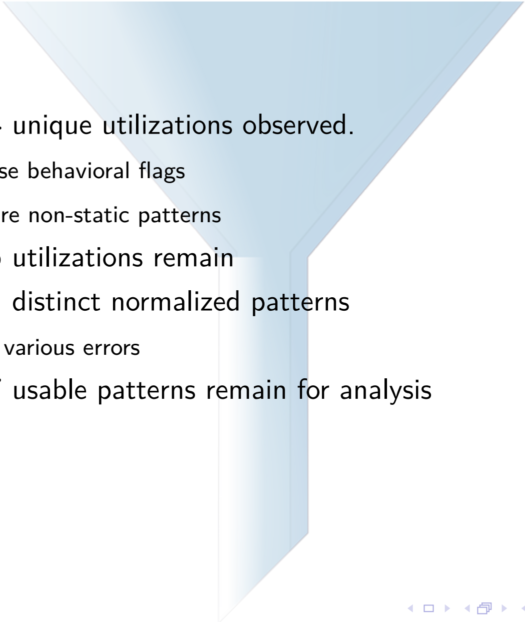
```
            function           pattern               flags
r1 = re.compile("(0|-?[1-9][0-9]*)$", re.MULTILINE)
```

function which function of the re module is called?

pattern string used to specify regex behavior

flags modifies the regex engine

Filtering utilizations and patterns



53,894 unique utilizations observed.

12.7% use behavioral flags

6.5% were non-static patterns

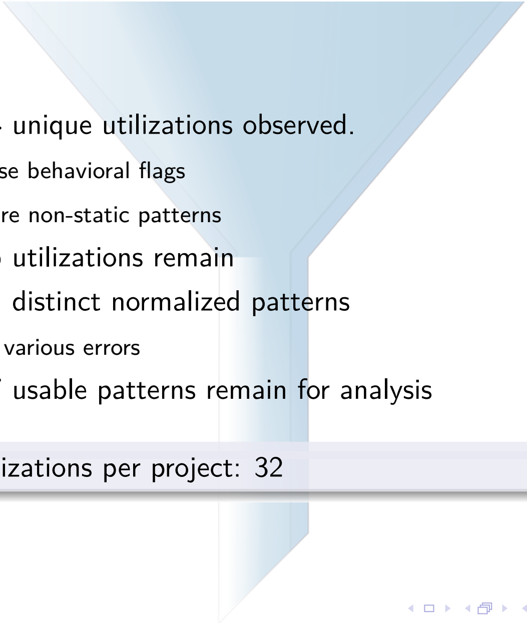
43,525 utilizations remain

13,711 distinct normalized patterns

114 had various errors

13,597 usable patterns remain for analysis

Filtering utilizations and patterns



53,894 unique utilizations observed.

12.7% use behavioral flags

6.5% were non-static patterns

43,525 utilizations remain

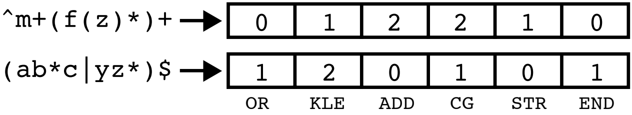
13,711 distinct normalized patterns

114 had various errors

13,597 usable patterns remain for analysis

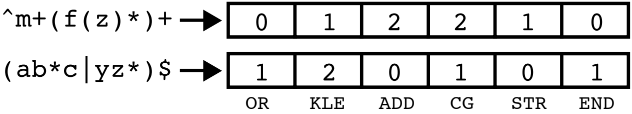
Average utilizations per project: 32

PCRE Parsing Patterns



	OR	KLE	ADD	CG	STR	END
`^m+(f(z)*)+`	0	1	2	2	1	0
`(ab*c|yz*)$`	1	2	0	1	0	1

PCRE Parsing Patterns



	OR	KLE	ADD	CG	STR	END
`^m+(f(z)*)+` →	0	1	2	2	1	0
`(ab*c|yz*)$` →	1	2	0	1	0	1

All Python features are recognizable by PCRE

Feature statistics - Top 8

Rank	Code	Example	% Projects	NProjects	NFiles	NPatterns	MaxTokens
1	ADD	`z+`	73.2	1,204	9,165	6,003	30
2	CG	`(caught)`	72.6	1,194	9,559	7,130	17
3	KLE	`.*`	66.8	1,099	8,163	6,017	50
4	CCC	`[aeiou]`	62.4	1,026	7,648	4,468	42
5	ANY	`.`	61.1	1,005	6,277	4,657	60
6	RNG	`[a-z]`	51.6	848	5,092	2,631	50
7	STR	`^`	51.4	846	5,458	3,563	12
8	END	`$`	50.3	827	5,393	3,169	12

Regex research tools

- Remember that we wanted to write a tool to support regex creation?

Regex research tools

- Remember that we wanted to write a tool to support regex creation?
- But regex feature usage references were missing.

Regex research tools

- Remember that we wanted to write a tool to support regex creation?
- But regex feature usage references were missing.
- So,

Regex research tools

- Remember that we wanted to write a tool to support regex creation?
- But regex feature usage references were missing.
- So,

How well do regex research tools meet the needs of developers? (Hampi, Rex, RE2, brics)

Which features are supported by analysis tools?

Rank	Code	Example	Brics	Hampi	Rex	RE2
1	ADD	z+	●	●	●	●
2	CG	(caught)	●	●	●	●
3	KLE	.*	●	●	●	●
4	CCC	[aeiou]	●	●	●	●
5	ANY	.	●	●	●	●
6	RNG	[a-z]	●	●	●	●
7	STR	^	🔴	●	●	●
8	END	$	🔴	●	●	●
9	NCCC	[^qwxf]	●	●	●	●
10	WSP	\s	🔴	●	●	●
11	OR	a|b	●	●	●	●
12	DEC	\d	🔴	●	●	●
13	WRD	\w	🔴	●	●	●
14	QST	z?	●	●	●	●
15	LZY	z+?	🔴	●	🔴	●
16	NCG	a(?:b)c	🔴	●	🔴	●
17	PNG	(?P<name>x)	🔴	●	🔴	🔴

Rank	Code	Example	Brics	Hampi	Rex	RE2
18	SNG	z{8}	●	●	●	●
19	NWSP	\S	🔴	●	●	●
20	DBB	z{3,8}	●	●	●	●
21	NLKA	a(?!yz)	🔴	🔴	🔴	🔴
22	WNW	\b	🔴	🔴	🔴	●
23	NWRD	\W	🔴	●	●	●
24	LWB	z{15,}	●	●	●	●
25	LKA	a(?=bc)	🔴	🔴	🔴	🔴
26	OPT	(?i)CasE	🔴	●	🔴	●
27	NLKB	(?<!x)yz	🔴	🔴	🔴	🔴
28	LKB	(?<=a)bc	🔴	🔴	🔴	🔴
29	ENDZ	\Z	🔴	🔴	🔴	●
30	BKR	\1	🔴	🔴	🔴	🔴
31	NDEC	\D	🔴	●	●	●
32	BKRN	\g<name>	🔴	●	🔴	🔴
33	VWSP	\v	🔴	🔴	●	●
34	NWNW	\B	🔴	🔴	🔴	●

Notable Observations

- Regexes are common, but not everywhere (42% of projects)
- Current regex research tools cover the most common features

Notable Observations

- Regexes are common, but not everywhere (42% of projects)
- Current regex research tools cover the most common features

but....

Notable Observations

- Regexes are common, but not everywhere (42% of projects)
- Current regex research tools cover the most common features

but....

What are the regexes doing?

How to find common behaviors?

1. ~~thorough inspection of 53K utilizations~~
2. ~~cluster by syntactic similarity like Jaccard or longest substring~~
3. ~~formal analytical subsumption, no sufficient tools at the moment~~
4. Chosen technique: cluster by behavioral similarity using Rex

Example

A `(ab*c|yz*)$`

- abbbbbbbc
- y
- abcy
- pac
- abcyzzz

B `(ab*c|yz*)`

- y
- abc
- abcy
- abcccc
- yxw

Example

A `(ab*c|yz*)$`

- abbbbbbbc
- y
- abcy
- pac
- abcyzzz

B `(ab*c|yz*)`

- y
- abc
- abcy
- abcccc
- yxw

A matches $3/5 =$ 60% of B's strings

Example

A `(ab*c|yz*)$`

- abbbbbbbc
- y
- abcy
- pac
- abcyzzz

A matches $3/5 =$ 60% of B's strings

B `(ab*c|yz*)`

- y
- abc
- abcy
- abcccc
- yxw

B matches $5/5 =$ 100% of A's strings

Example

A `(ab*c|yz*)$`

- abbbbbbbc
- y
- abcy
- pac
- abcyzzz

A matches $3/5 =$ 60% of B's strings

B `(ab*c|yz*)`

- y
- abc
- abcy
- abcccc
- yxw

B matches $5/5 =$ 100% of A's strings

A and B are 80% similar

Similarity Matrix → MCL

```
Pattern A matches 100/100 of A's strings
Pattern B matches  90/100 of A's strings
Pattern A matches  50/100 of B's strings
Pattern B matches 100/100 of B's strings
```

	A	B
A	1.0	0.9
B	0.5	1.0

	A	B	C	D
A	1.0	0.0	0.9	0.0
B	0.2	1.0	0.8	0.7
C	0.6	0.8	1.0	0.2
D	0.0	0.6	0.1	1.0

→

	A	B	C	D
A	1.0			
B	0.1	1.0		
C	0.75	0.8	1.0	
D	0.0	0.65	0.15	1.0

Rex generates
400 strings for each regex.
Convert scores to half-matrix for
MCL

Clustering Results

Example Cluster

Index	Pattern	NProjects	Index	Pattern	NProjects
1	`\s*([^: ]*)\s*:(.*)`	9	7	`[:]`	6
2	`:+`	8	8	`([^:]+):(.*)`	6
3	`(:)`	8	9	`\s*:\s*`	4
4	`(:+)`	8	10	`\:`	2
5	`(:)(:*)`	8	11	`^([^:]*):[^:]*$`	2
6	`^([^:]*): *(.*)`	8	12	`^[^:]*:([^:]*)$`	2

From 2,871 distinct regexes

186 clusters where size ≥ 2

2,042 unclustered regexes

Six Categories Of Clusters

Category	Clusters	Patterns	Projects	% Projects
Multi Matches	21	237	295	40%
Specific Char	17	103	184	25%
Anchored Patterns	20	85	141	19%
Two or More Chars	16	40	120	16%
Content of Parens	10	46	111	15%
Code Search	15	27	92	13%

Multi Matches `(\s)` , `,|;`

Specific Char `:+` , `}` , `%`

Anchored Patterns `^[-_A-Za-z0-9]+$`

Two Or More Chars `@[a-z]+`

Content of Parens `<(.+)>` , `<[^>]*?>`

Code Search `.*rlen=([0-9]+)`

Notable Observations

- Finding a specific character is quite common, 25% of projects (in contrast with survey)
- Regexes are often used to parse source code!

Notable Observations

- Finding a specific character is quite common, 25% of projects (in contrast with survey)
- Regexes are often used to parse source code!

but....

Notable Observations

- Finding a specific character is quite common, 25% of projects (in contrast with survey)
- Regexes are often used to parse source code!

but....

What's the bigger picture?

Opportunities for future work!

Opportunities for future work!

Refactoring Regular Expressions

Regexes are hard to read. Could refactoring help?

Opportunities for future work!

Refactoring Regular Expressions

Regexes are hard to read. Could refactoring help?

Migration Support for Developers

Supported regex features are different among languages.

Opportunities for future work!

Refactoring Regular Expressions

Regexes are hard to read. Could refactoring help?

Migration Support for Developers

Supported regex features are different among languages.

Better Similarity Metrics

Our similarity metrics are empirical, can we do it analytically?

Opportunities for future work!

Refactoring Regular Expressions

Regexes are hard to read. Could refactoring help?

Migration Support for Developers

Supported regex features are different among languages.

Better Similarity Metrics

Our similarity metrics are empirical, can we do it analytically?

Identifying Best Practices?

Could impact regex education and improve comprehension.

Recap

- Regexes are (sort of) everywhere! (42% of Python projects studied)

Recap

- Regexes are (sort of) everywhere! (42% of Python projects studied)
- Everyone writes regexes! (50% of surveyed developers write them weekly)

Recap

- Regexes are (sort of) everywhere! (42% of Python projects studied)
- Everyone writes regexes! (50% of surveyed developers write them weekly)
- Regexes are hard to read/write! (this was reported as a pain point)

Recap

- Regexes are (sort of) everywhere! (42% of Python projects studied)
- Everyone writes regexes! (50% of surveyed developers write them weekly)
- Regexes are hard to read/write! (this was reported as a pain point)

also...

- Current tools support most of the most common features
- Regexes are often used for code search
- Many opportunities for future work!

Questions?

Katie Stolee – ktstolee@ncsu.edu

Questions?

Katie Stolee – ktstolee@ncsu.edu

(psst! Graduate students! I'm actively recruiting for my research lab!)