Language & Technology

Lecture 3: String Matching

Thomas Graf

Stony Brook University
lin120@thomasgraf.net

I had a Very Fun Saturday...

- **Task:**
  collect data on SBC-certified courses over last 5 years
- **Solution:**
  - Download all archived undergraduate bulletins.
  - For each course, extract which SBC requirements it satisfies.

But how do you do that?

- ≈3,500 registered courses
- course descriptions scattered across 140 files per semester
- For a 5-year period, that's over **35,000** course descriptions scattered over **1,400 files**!

I had a Very Fun Saturday...

- **Task:**
  collect data on SBC-certified courses over last 5 years
- **Solution:**
  - Download all archived undergraduate bulletins.
  - For each course, extract which SBC requirements it satisfies.

But how do you do that?

- $\approx$3,500 registered courses
- course descriptions scattered across 140 files per semester
- For a 5-year period, that's over **35,000** course descriptions scattered over **1,400 files**!

The Final Solution

- This took quite a bit of data massaging with Python.
- But the central step is condensing course descriptions into a list of the form

```
[program, course_number, course_name, SBCs]
```

- And here is the central piece of code that does the trick:

```
re.sub(r'^.*?id="(\w+)".*?<h3>.*?:\s*(.*?)</h3>.*?<p>(.*?)</p>(.*)',
       r"\1|\2|\3|\4",
       course_description)
```

- What the heck is that? It's the miraculous (and highly illegible) world of **regular expressions**!

The Final Solution

- This took quite a bit of data massaging with Python.
- But the central step is condensing course descriptions into a list of the form

```
[program, course_number, course_name, SBCs]
```

- And here is the central piece of code that does the trick:

```
re.sub(r'^.*?id="(\w+)".*?<h3>.*?:\s*(.*?)</h3>.*?<p>(.*?)</p>(.*)',
       r"\1|\2|\3|\4",
       course_description)
```

- What the heck is that? It's the miraculous (and highly illegible) world of **regular expressions**!

The Final Solution

- This took quite a bit of data massaging with Python.
- But the central step is condensing course descriptions into a list of the form

```
[program, course_number, course_name, SBCs]
```

- And here is the central piece of code that does the trick:

```
re.sub(r'^.*?id="(\w+)".*?<h3>.*?:\s*(.*?)</h3>.*?<p>(.*?)</p>(.*)',
       r"\1|\2|\3|\4",
       course_description)
```

- What the heck is that? It's the miraculous (and highly illegible) world of **regular expressions**!

Let's Take a Step Back

- Regular expressions are about matching patterns in a string.
- This is an essential technique for language technology.
- But regular expressions even reveal a metaphysical truth about knowledge.
- Let's look at both, starting with the big picture.

Some Terminology

- An **alphabet** is a fixed collection of symbols.

Example

- Latin alphabet plus punctuation symbols and space
- $\clubsuit$, $\diamond$, $\blacktriangleleft$
- 0 and 1
- cytosine, guanine, adenine, thymine

- A **string** is a sequence of finitely many symbols drawn from some alphabet Σ (an uppercase sigma).
- The collection of all strings over Σ is called $\mathbf{\Sigma}^*$ ("Sigma star").

A Common Alphabet for Computers: ASCII

The ASCII alphabet contains 128 characters:

1. lowercase letters
   a b c . . . z
2. uppercase letters
   A B C . . . Z
3. punctuation
   . ! ? , : ; -
4. whitespace
   space tabulator linebreak
5. parenthesis
   ( ) [ ] { }
6. special characters
   @ # $ + . . .
7. some weird stuff
   Vertical tab, Form Feed

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

The Vastness of Σ^*

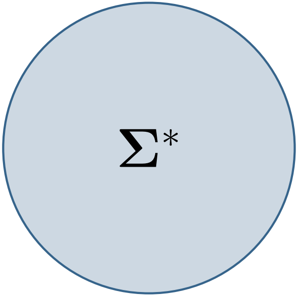
- Suppose Σ is the ASCII alphabet.
- Then what does Σ^* contain?

The Vastness of Σ^*

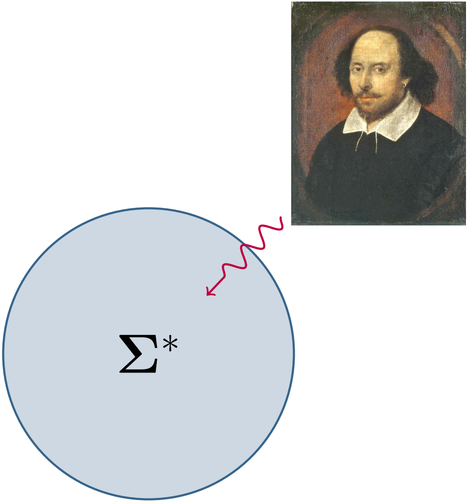
- Suppose Σ is the ASCII alphabet.
- Then what does Σ^* contain?

Everything!

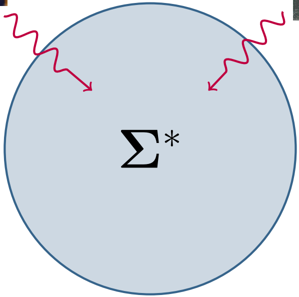
Elements of Σ^*



Elements of Σ^*



Elements of Σ^*



Elements of Σ^*

$$\Sigma^*$$

Σ^* has all the answers...

Σ^* contains everything that can be expressed with the chosen alphabet Σ.

- **ASCII**
  - collected works of Shakespeare
  - all tweets that Donald Trump deleted halfway through
  - the funniest joke never told
  - the list of your previous boy/girlfriends
- **ASCII + mathematical symbols**
  - all truths of mathematics
  - all numbers that can be read as an English word (80085)
- **cytosine, guanine, adenine, thymine**
  - the genomes of all humans that were abducted by aliens

It seems like Σ^* is the key to **omniscience**.

And we can generate Σ^*...

- It is very easy to write a generator for Σ^*.

```
# define some alphabet
alphabet = [a, b, c, # and so on

# add a special symbol, e.g. ß, to the alphabet
# to separate distinct members of Sigma*
list.append(alphabet, "ß")

# then start generating indefinitely
while True:
    print(random.choice(alphabet), end="")
```

Example Output After Some Time

```
"aas/ fk-j 23 819ßakerß555The answer to the next quiz isß/asd;k"
```

- Keep the code running forever, and it will produce everything that could ever be written, said, or thought.

And we can generate Σ^*...

- It is very easy to write a generator for Σ^*.

```
# define some alphabet
alphabet = [a, b, c, # and so on

# add a special symbol, e.g. ß, to the alphabet
# to separate distinct members of Sigma*
list.append(alphabet, "ß")

# then start generating indefinitely
while True:
    print(random.choice(alphabet), end="")
```

Example Output After Some Time

```
"aas/ fk-j 23 819ßakerß555The answer to the next quiz isß/asd;k"
```

- Keep the code running forever, and it will produce everything that could ever be written, said, or thought.

And we can generate Σ^*...

- It is very easy to write a generator for Σ^*.

```
# define some alphabet
alphabet = [a, b, c, # and so on

# add a special symbol, e.g. ß, to the alphabet
# to separate distinct members of Sigma*
list.append(alphabet, "ß")

# then start generating indefinitely
while True:
    print(random.choice(alphabet), end="")
```

Example Output After Some Time

```
"aas/ fk-j 23 819ßakerß555The answer to the next quiz isß/asd;k"
```

- Keep the code running forever, and it will produce everything that could ever be written, said, or thought.

The Proverbial Monkey on a Typewriter

- **Problem**: our generator mostly produces gibberish

The Moral of the Story

- Σ^* contains all truths.
- But the truth is drowned out among the noise.
- True knowledge is the ability to filter out the noise:
  1. Know **what to look for**.
  2. Know **how to look for it**.

A Friendly Pointer

- Jorge Luis Borges' short story *The Library of Babel*
- The library exists!
  Try it online at `https://libraryofbabel.info/`

Back to Regular Expressions

- What does all of this have to do with regular expressions?
- Regular expressions are our tool for getting rid of the noise.
- Just like Σ^* is full of irrelevant noise,
  any given string may be full of irrelevant stuff.
- With regular expressions, we can pick out the parts
  that we care about and forget about the rest.

Basic Building Blocks of Regular Expressions

The syntax of regular expressions varies slightly between implementations. We'll follow the Python conventions:

Pattern	**Match**
string	string
.	any single character, including whitespace
^	beginning of line
$	end of line
[x,y, . . .]	characters x, y, . . .
[x-y]	all characters from x to y
(x | y)	strings matching x or y
x$\{m\}$	exactly m instances of x
x$\{m,\}$	at least m instances of x
x$\{,n\}$	at most n instances of x
x$\{m,n\}$	between m and n instances of x
x?	same as x$\{0,1\}$ $\Rightarrow$ x is optional
x+	same as x$\{1,\}$ $\Rightarrow$ one or more x
x$*$	same as x$\{0,\}$ $\Rightarrow$ zero or more x

Examples of Regular Expressions

Below are a few regexes, and for each regex some English words that match the described pattern.

Regex	Some Representative Matches
`test`	
`tes?t`	
`tes?t?`	
`t(es)?t`	
`^test`	
`^te+s?t`	
`^.?t.*e+s?t`	
`^.?t.*e+s?t$`	
`^te+[a-c].*s$`	
`^te+([a-c]|st?).*s$`	

Examples of Regular Expressions

Below are a few regexes, and for each regex some English words that match the described pattern.

Regex	Some Representative Matches
`test`	test, testing, protests, vastest
`tes?t`	
`tes?t?`	
`t(es)?t`	
`^test`	
`^te+s?t`	
`^.?t.*e+s?t`	
`^.?t.*e+s?t$`	
`^te+[a-c].*s$`	
`^te+([a-c]|st?).*s$`	

Examples of Regular Expressions

Below are a few regexes, and for each regex some English words that match the described pattern.

<table>
<tr><th>Regex</th><th>Some Representative Matches</th></tr>
<tr><td><code>test</code></td><td>test, testing, protests, vastest</td></tr>
<tr><td><code>tes?t</code></td><td>test, testing, protests, vastest, untether</td></tr>
<tr><td><code>tes?t?</code></td><td></td></tr>
<tr><td><code>t(es)?t</code></td><td></td></tr>
<tr><td><code>^test</code></td><td></td></tr>
<tr><td><code>^te+s?t</code></td><td></td></tr>
<tr><td><code>^.?t.*e+s?t</code></td><td></td></tr>
<tr><td><code>^.?t.*e+s?t$</code></td><td></td></tr>
<tr><td><code>^te+[a-c].*s$</code></td><td></td></tr>
<tr><td><code>^te+([a-c]|st?).*s$</code></td><td></td></tr>
</table>

Examples of Regular Expressions

Below are a few regexes, and for each regex some English words that match the described pattern.

Regex	Some Representative Matches
`test`	test, testing, protests, vastest
`tes?t`	test, testing, protests, vastest, untether
`tes?t?`	test, testing, winter, zygotes, stethoscope
`t(es)?t`	
`^test`	
`^te+s?t`	
`^.?t.*e+s?t`	
`^.?t.*e+s?t$`	
`^te+[a-c].*s$`	
`^te+([a-c]|st?).*s$`	

Examples of Regular Expressions

Below are a few regexes, and for each regex some English words that match the described pattern.

Regex	**Some Representative Matches**
`test`	test, testing, protests, vastest
`tes?t`	test, testing, protests, vastest, untether
`tes?t?`	test, testing, winter, zygotes, stethoscope
`t(es)?t`	test, testing, vastest, wettest, better
`^test`	
`^te+s?t`	
`^.?t.*e+s?t`	
`^.?t.*e+s?t$`	
`^te+[a-c].*s$`	
`^te+([a-c]|st?).*s$`	

Examples of Regular Expressions

Below are a few regexes, and for each regex some English words that match the described pattern.

Regex	Some Representative Matches
`test`	test, testing, protests, vastest
`tes?t`	test, testing, protests, vastest, untether
`tes?t?`	test, testing, winter, zygotes, stethoscope
`t(es)?t`	test, testing, vastest, wettest, better
`^test`	test, testing, testify
`^te+s?t`	
`^.?t.*e+s?t`	
`^.?t.*e+s?t$`	
`^te+[a-c].*s$`	
`^te+([a-c]|st?).*s$`	

Examples of Regular Expressions

Below are a few regexes, and for each regex some English words that match the described pattern.

Regex	Some Representative Matches
`test`	test, testing, protests, vastest
`tes?t`	test, testing, protests, vastest, untether
`tes?t?`	test, testing, winter, zygotes, stethoscope
`t(es)?t`	test, testing, vastest, wettest, better
`^test`	test, testing, testify
`^te+s?t`	
`^.?t.*e+s?t`	
`^.?t.*e+s?t$`	
`^te+[a-c].*s$`	
`^te+([a-c]|st?).*s$`	

Examples of Regular Expressions

Below are a few regexes, and for each regex some English words that match the described pattern.

Regex	**Some Representative Matches**
`test`	test, testing, protests, vastest
`tes?t`	test, testing, protests, vastest, untether
`tes?t?`	test, testing, winter, zygotes, stethoscope
`t(es)?t`	test, testing, vastest, wettest, better
`^test`	test, testing, testify
`^te+s?t`	test, testing, testify, teet
`^.?t.*e+s?t`	
`^.?t.*e+s?t$`	
`^te+[a-c].*s$`	
`^te+([a-c]|st?).*s$`	

Examples of Regular Expressions

Below are a few regexes, and for each regex some English words that match the described pattern.

Regex	**Some Representative Matches**
`test`	test, testing, protests, vastest
`tes?t`	test, testing, protests, vastest, untether
`tes?t?`	test, testing, winter, zygotes, stethoscope
`t(es)?t`	test, testing, vastest, wettest, better
`^test`	test, testing, testify
`^te+s?t`	test, testing, testify, teet
`^.?t.*e+s?t`	test, testing, testify, teet, street, strangest
`^.?t.*e+s?t$`	
`^te+[a-c].*s$`	
`^te+([a-c]|st?).*s$`	

Examples of Regular Expressions

Below are a few regexes, and for each regex some English words that match the described pattern.

Regex	Some Representative Matches
`test`	test, testing, protests, vastest
`tes?t`	test, testing, protests, vastest, untether
`tes?t?`	test, testing, winter, zygotes, stethoscope
`t(es)?t`	test, testing, vastest, wettest, better
`^test`	test, testing, testify
`^te+s?t`	test, testing, testify, teet
`^.?t.*e+s?t`	test, testing, testify, teet, street, strangest
`^.?t.*e+s?t$`	test, teet, street, strangest
`^te+[a-c].*s$`	
`^te+([a-c]|st?).*s$`	

Examples of Regular Expressions

Below are a few regexes, and for each regex some English words that match the described pattern.

Regex	**Some Representative Matches**
`test`	test, testing, protests, vastest
`tes?t`	test, testing, protests, vastest, untether
`tes?t?`	test, testing, winter, zygotes, stethoscope
`t(es)?t`	test, testing, vastest, wettest, better
`^test`	test, testing, testify
`^te+s?t`	test, testing, testify, teet
`^.?t.*e+s?t`	test, testing, testify, teet, street, strangest
`^.?t.*e+s?t$`	test, teet, street, strangest
`^te+[a-c].*s$`	tea's, teachers, teazels, technocrats
`^te+([a-c]|st?).*s$`	

Examples of Regular Expressions

Below are a few regexes, and for each regex some English words that match the described pattern.

Regex	Some Representative Matches
`test`	test, testing, protests, vastest
`tes?t`	test, testing, protests, vastest, untether
`tes?t?`	test, testing, winter, zygotes, stethoscope
`t(es)?t`	test, testing, vastest, wettest, better
`^test`	test, testing, testify
`^te+s?t`	test, testing, testify, teet
`^.?t.*e+s?t`	test, testing, testify, teet, street, strangest
`^.?t.*e+s?t$`	test, teet, street, strangest
`^te+[a-c].*s$`	tea's, teachers, teazels, technocrats
`^te+([a-c]|st?).*s$`	tests, testing, tea's, technocracts

So What are Regexes Good For?

- Regular expressions are not specific to language technology.
  - text search
  - search and replace
  - renaming files
  - syntax highlighting for programmers (like in Jupyter notebook)
- But they are part of tons of language technology:
  - chatbots (see the homeworks)
  - data analysis (discussed later in semester)
  - simple grammar checker

Regexes for Grammar Checking

- A **real-word error** is a spelling error where the word is spelled incorrectly given its context.
- Example: *Their is a man in the garden*.
- can be detected with regular expressions

```
r"[Tt]heir (is|are|may|must|seems? )"
```

Another Grammar Checking Example

Number agreement between subject and verb:

(1) a. (All) (the) neighbors of Bill are always annoying.

b. * (All) (the) neighbors of Bill is always annoying.

Simplified Regex for Finding Agreement Error

```
r"^(All )?[Tt]he [A-Za-z]+s( of [A-Za-z]+)? is"
```

- In practice, regexes are too clunky for fully adequate grammar checking.
- But regexes are part of many grammar checkers.

Quick Break: Another Practice Session

For each regex, say whether *Sue left...* is matched by it.

```
r"^s.+t"
```

```
r"^S.+t"
```

```
r"^S.+t$"
```

```
r"^S.+t\.$"
```

```
r"^S.+t\.*$"
```

```
r"^S.+(p|r|f).*t\.*$"
```

Quick Break: Another Practice Session

For each regex, say whether *Sue left...* is matched by it.

```
r"^s.+t"
```

```
r"^S.+t"
```

```
r"^S.+t$"
```

```
r"^S.+t\.$"
```

```
r"^S.+t\.*$"
```

```
r"^S.+(p|r|f).*t\.*$"
```

Quick Break: Another Practice Session

For each regex, say whether *Sue left...* is matched by it.

```
r"^s.+t"
```

```
r"^S.+t"
```

```
r"^S.+t$"
```

```
r"^S.+t\.$"
```

```
r"^S.+t\.*$"
```

```
r"^S.+(p|r|f).*t\.*$"
```

Quick Break: Another Practice Session

For each regex, say whether *Sue left...* is matched by it.

```
r"^s.+t"
```

```
r"^S.+t"
```

```
r"^S.+t$"
```

```
r"^S.+t\.$"
```

```
r"^S.+t\.*$"
```

```
r"^S.+(p|r|f).*t\.*$"
```

Quick Break: Another Practice Session

For each regex, say whether *Sue left...* is matched by it.

```
r"^s.+t"
```

```
r"^S.+t"
```

```
r"^S.+t$"
```

```
r"^S.+t\.$"
```

```
r"^S.+t\.*$"
```

```
r"^S.+(p|r|f).*t\.*$"
```

Quick Break: Another Practice Session

For each regex, say whether *Sue left...* is matched by it.

```
r"^s.+t"
```

```
r"^S.+t"
```

```
r"^S.+t$"
```

```
r"^S.+t\.$"
```

```
r"^S.+t\.*$"
```

```
r"^S.+(p|r|f).*t\.*$"
```

Quick Break: Another Practice Session

For each regex, say whether *Sue left...* is matched by it.

```
r"^s.+t"
```

```
r"^S.+t"
```

```
r"^S.+t$"
```

```
r"^S.+t\.$"
```

```
r"^S.+t\.*$"
```

```
r"^S.+(p|r|f).*t\.*$"
```

Quick Break: Another Practice Session

For each regex, say whether *Sue left...* is matched by it.

```
r"^s.+t"
```

```
r"^S.+t"
```

```
r"^S.+t$"
```

```
r"^S.+t\.$"
```

```
r"^S.+t\.*$"
```

```
r"^S.+(p|r|f).*t\.*$"
```

Answers

1. No 2. No 3. No 4. No 5. Yes 6. Yes

Additional Regex Tricks: Special Characters and Negation

- A text may contain tabs and new lines, which we can't type directly in regexes.
- Instead, we have to use the special characters `\t` and `\n`.

```
# match comma, semicolon or hyphen before a linebreak
r"[,;-]\n"
```

- Sometimes, we want to say "do **not** match x, y, or z".
- This is written `[^xyz]`.

```
# match comma, semicolon or hyphen NOT before a linebreak
r"[,;-][^\n]"
```

Additional Regex Tricks: Classes

We can already define lists of characters,
but sometimes this can be cumbersome.

```
# a regex for matching any word or variable name
r"[A-Za-z0-9_]+"
```

Classes are shorthands for specific lists.

Class	Equivalent to	Mnemonic
`\w`	`[A-Za-z0-9_]`	**w**ord
`\W`	`[^A-Za-z0-9_]`	**NOT w**ord
`\d`	`[0-9]`	**d**igit
`\D`	`[^0-9]`	**NOT d**igit
`\s`	`[ \t\n]`	white**s**pace
`\S`	`[^ \t\n]`	**NOT** white**s**pace

Some Examples

1. Matching all words, and nothing else
(this will be really important for us in the next lecture)

```
r"\w+"
```

2. Matching all AD years

```
r"\d{1,4}"
```

3. Finding two words with arbitrary amount of whitespace between them

```
r"\w+\s+\w+"
```

The Final Trick: Backreferences

- The brackets (and) also define groups.
- For example, the **backreference** \2 refers to the 2nd group.

```python
# convert dates from month/day/full_year to full_year-month-day
re.sub(r"(\w+)/(\d{,2})/(\d{4})",
       r"\3-\1-\2",
       string)
```

- Backreferences allow chatbots to reuse parts of the user input.

Tips for Reading and Writing Regular Expressions

- **Practice, practice, practice!**
  Regular expressions take a while to get used to. Practice on
  - Pythex: `https://pythex.org`
  - RegexOne: `https://regexone.com/lesson/introduction_abcs`
- **Don't panic!**
  Regexes look confusing, but just work your way trough them left-to-right and you'll soon know what's going on.
- **It's the patterns, stupid!**
  If you want to clean up a string with regexes, think about what exactly the pattern is that you're trying to extract. If you can phrase it as something like "all the stuff between the first vowel and the third consonant", you are already halfway done with your regex.

Summary

- Regular expressions are an essential for pattern matching.
  - clean up data
  - modify and recycle user input
  - grammar checking
  - and much more
- At this point they will still feel strange to you, but by the end of the semester you'll be fairly comfortable with them.