

Regex Basics and Sanity

By the end hopefully we have both...

Ok, so lets break into this easy and test out using literals. This is exactly as it sounds and matches literally what we type (to a degree as we will see later)

Here we have matched “boo”

The screenshot shows a regular expression testing interface. The top bar says "REGULAR EXPRESSION" and has a search field containing "/ boo". To the right, it says "3 matches (12 steps, 0.1ms)" and has buttons for "/ gm" and a copy icon. Below this is a "TEST STRING" section containing the text "boo boomerang taboo". Three red arrows point from the text to the first three words. To the right is a "MATCH INFORMATION" section with three entries:

Match	Range	Value
Match 1	0-3	boo
Match 2	4-7	boo
Match 3	16-19	boo

A red arrow points to the second entry, "Match 2".

Now, lets do it again but with numbers. Notice, just like “boo”, it finds it anywhere and matches.

Here we have matched “1000”

The screenshot shows a regular expression testing interface. The regular expression input field contains ': / 1000'. The test string is:
boo
boomerang
taboo
1000
1000000
0101000010

The 'MATCH INFORMATION' panel on the right lists three matches:

Match	Range	Value
Match 1	20-24	1000
Match 2	25-29	1000
Match 3	36-40	1000

Red arrows point from the text 'Here we have matched "1000"' to the second occurrence of '1000' in the test string and to the second match entry in the 'MATCH INFORMATION' table.

Lets now match only rows that start with “boo”.

We use “^” to say
match what's next but
it must be at the
beginning.

The screenshot shows a regular expression tester interface. The regular expression input field contains the pattern `^ boo`. The test string list includes `boo`, `boomerang`, `taboo`, `1000`, `1000000`, and `0101000010`. The match information panel shows two matches: Match 1 at index 0-3 with the value `boo`, and Match 2 at index 4-7 with the value `boo`. Red arrows point from the explanatory text to the '^' character in the regex input field, to the first two items in the test string list, and to the first match entry in the match information panel.

Lets do it again and switch to numbers this time.

Again, use “^” and see it finds 1000 only when its the beginning.

The screenshot shows a regular expression testing interface. The regular expression input field contains '^1000'. The test string input field contains several lines of text: 'boo', 'boomerang', 'taboo', '1000', '1000000', and '0101000010'. The match information panel on the right shows two matches:

Match	Range	Value
Match 1	20-24	1000
Match 2	25-29	1000

Red arrows point from the '^' in the regex input to the first '1000' in the test string, and from the second '1000' in the test string to the 'Match 2' row in the match information table.

This time lets add “\$” to the end and see what we get. Notice, we only have one match now. That is because \$ says match whatever is before but only at the end.

So, `^boo$` says
match me but only if I
start and end the row
with boo.

The screenshot shows a regular expression tester interface. At the top, the regular expression input field contains `/^boo$/`. To the right of the input field, there is a status bar that says "1 match (14 steps, 0.1ms)". Below the input field is a "TEST STRING" section containing the following list of words:
boo
boomerang
taboo
1000
1000000
0101000010

On the right side of the interface is a "MATCH INFORMATION" panel. It displays "Match 1" with the range "0-3" and the matched string "boo". A red arrow points from the explanatory text above to the "boo" entry in the test string list. Another red arrow points from the explanatory text to the "boo" entry in the match information panel.

Lets check on numbers and see if the rules change.

Looks good and again 1 match because it must start and end with 1000

The screenshot shows a regular expression tester interface. At the top, the regular expression field contains `/^1000$/`. To the right of the field, there is a green button that says "1 match (22 steps, 0.1ms)". Below the expression field is a "TEST STRING" section containing the following list of strings:

- boo
- boomerang
- taboo
- 1000
- 1000000
- 0101000010

Red arrows point from the text "1000" in the "TEST STRING" list to the regular expression field and to the "Match 1" entry in the "MATCH INFORMATION" panel. The "MATCH INFORMATION" panel shows a table with one row:

Match 1	20-24	1000

A red arrow points to the value "1000" in the third column of the table.

Ok, lets kick it up a notch...

Lets try part of an address but don't use literals. There are many ways to do this and here are a few examples. Notice how we continue to refine what took 35 steps to start ended at only taking 3.

Helpful predefined classes

\d – digits

\s – space

\w – letter and digit

\D – anything not a digit

So, I liked how strong \D was and for the next string I'm going to jump right into using it.

First I'll input \D and note 33 matches

Then \d and 5 matches

This helps because I don't want to count characters so I now know exactly what numbers to use to select the entire string

The screenshot shows a regular expression testing interface with three separate sections:

- REGULAR EXPRESSION:** `: / \D` (highlighted with a red arrow) → **33 matches (66 steps, 0.1ms)**
- REGULAR EXPRESSION:** `: / \d` → **5 matches (10 steps, 0.0ms)**
- REGULAR EXPRESSION:** `: / \D{33}\d{5}` (highlighted with a red arrow) → **1 match (3 steps, 0.0ms)**

TEST STRING: John•Q.•Adams,•Denver,•Colorado,•80123

MATCH INFORMATION:

Match 1	0-38	John•Q.•Adams,•Denver,•Colorado, •80123
---------	------	--

Fairly straight forward but again there are many ways to match.

Lets gets weird and try spelling variation selection. The key here is “?” which says match the previous item 0-1 times. Basically, I’m ok if its there or not and keep matching.

REGULAR EXPRESSION

2 matches (16 steps, 0.0ms)

:/ Flavou?r|

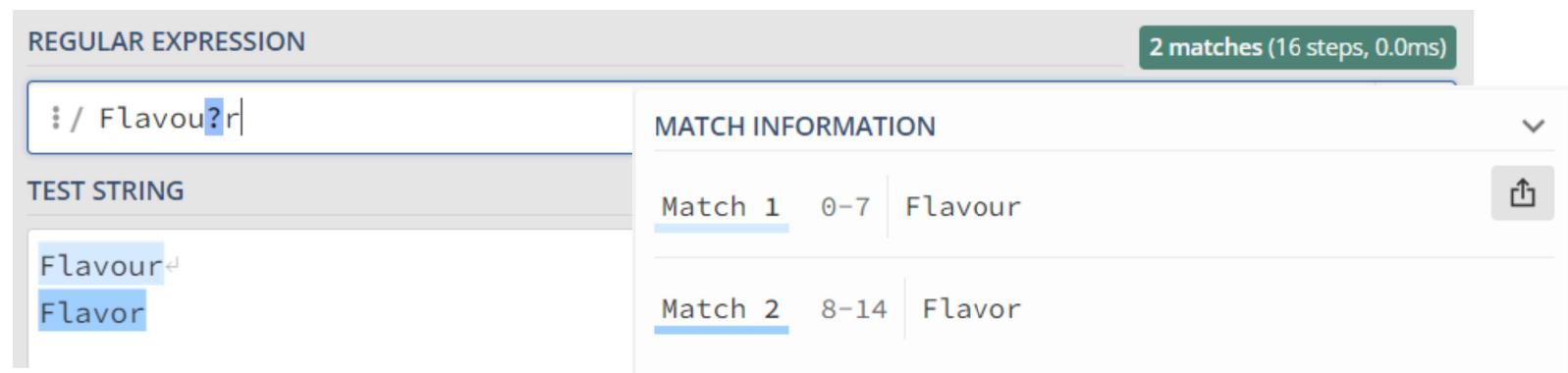
TEST STRING

Flavour
Flavor

MATCH INFORMATION

Match 1 0-7 Flavour

Match 2 8-14 Flavor



However, if we just want to match everything in one fell swoop I will introduce you to “+”. This says match the previous until you can't. Powerful but remember with great power comes great responsibility...

REGULAR EXPRESSION

:/ \D+

1 match (2 steps, 0.0ms)

/ gm

TEST STRING

Flavour[✉]
Flavor

MATCH INFORMATION

Match 1 0-14 Flavour[✉]
Flavor



Ok, next up lets create a character class. We do that with “[]” and below I said match anything but numbers.

The screenshot shows a regular expression testing interface. On the left, under 'REGULAR EXPRESSION', the pattern `:/ [^0-9]{41}/ gm` is entered. The result is '1 match (2 steps, 0.1ms)'. Below it, the 'TEST STRING' is `Today•we're•learning•regular•expressions.`. On the right, under 'MATCH INFORMATION', Match 1 is highlighted, showing the range 0-41 and the matched string `Today•we're•learning•regular•expressions.`.

Lets start to select individual strings now. Here we see each string ends (hint) in day. So, lets say I want to match any string that ends (\$) in day.

REGULAR EXPRESSION

3 matches (29 steps, 0.0ms)

/ .+day\$/ gm

TEST STRING

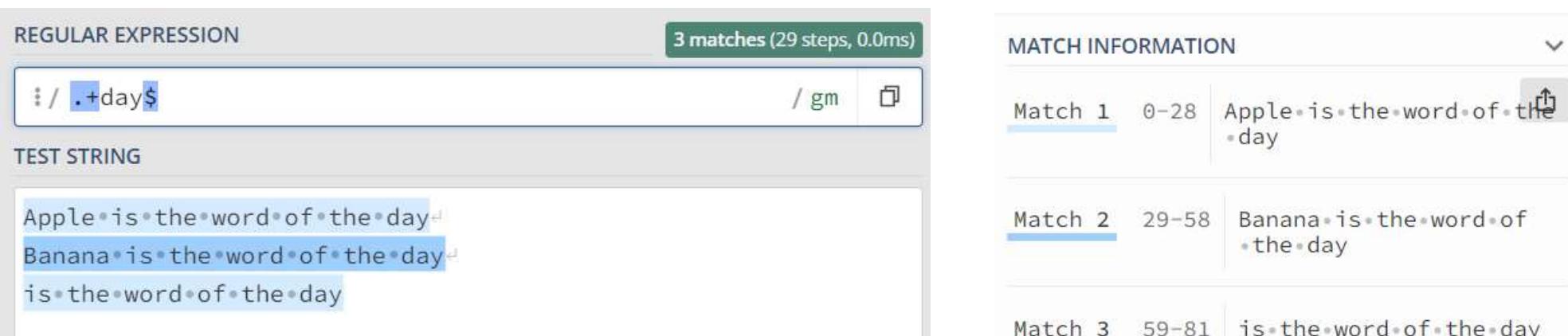
Apple•is•the•word•of•the•day
Banana•is•the•word•of•the•day
is•the•word•of•the•day

MATCH INFORMATION

Match 1 0-28 Apple•is•the•word•of•the•day

Match 2 29-58 Banana•is•the•word•of•the•day

Match 3 59-81 is•the•word•of•the•day



To select the entire string lets use \D+ and call it a day.

REGULAR EXPRESSION

1 match (2 steps, 0.0ms)

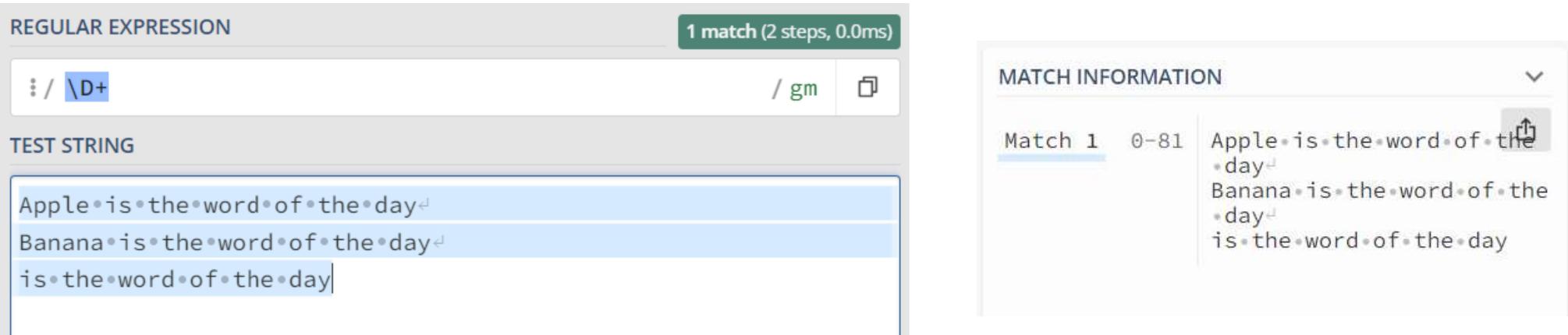
/ \D+ gm

TEST STRING

Apple•is•the•word•of•the•day
Banana•is•the•word•of•the•day
is•the•word•of•the•day

MATCH INFORMATION

Match 1 0-81 Apple•is•the•word•of•the•day
Banana•is•the•word•of•the•day
is•the•word•of•the•day



Now, lets start to match strings that you may find in a log. Sure we can use a classic `.+` but that has limitation. Lets think a bit more...

Ok, so I went with match anything (no numbers) at start and stop at “in” because I used it literally.

The screenshot shows a user interface for testing regular expressions against a test string. The test string contains three entries: "User•Tom•logged•in", "User•contoso\batman•logged•in", and "User•contoso\joker•logged•in". The first entry is highlighted with a blue selection bar. Below the test string, there are two sections for regular expressions. The top section shows a regular expression `:/ .+` with a green button indicating "3 matches (8 steps, 0.0ms)". The bottom section shows a regular expression `:/ ^\D*?in|` with a green button indicating "3 matches (86 steps, 0.1ms)". To the right of the test string, a "MATCH INFORMATION" panel lists three matches with their start and end indices and corresponding captured groups:

Match	Index Range	Captured Group
Match 1	0-18	User•Tom•logged•in
Match 2	19-48	User•contoso\batman•logged•in
Match 3	49-77	User•contoso\joker•logged•in

I wasn't happy so I had to try again. This time lets toss some groups into play. Sure, very similar to before but seeing it this way feels better. Lets keep learning and see if this develops.

TEST STRING

```
User•Tom•logged•in
User•contoso\batman•logged•in
User•contoso\joker•logged•in
```

REGULAR EXPRESSION

3 matches (60 steps, 0.0ms)

```
/(User)(.+)(in)
```

/ gm



MATCH INFORMATION

Match 1	0-18	User•Tom•logged•in
Group 1	0-4	User
Group 2	4-16	•Tom•logged•
Group 3	16-18	in
Match 2	19-48	User•contoso\batman •logged•in
Group 1	19-23	User
Group 2	23-46	•contoso\batman•logged•
Group 3	46-48	in
Match 3	49-77	User•contoso\joker •logged•in
Group 1	49-53	User
Group 2	53-75	•contoso\joker•logged•
Group 3	75-77	in

Ok, lets work on spelling variations again. This time lets build a character. As we learn, think about ways to apply in a work setting. Then go mess with it yourself.

The screenshot shows a user interface for testing regular expressions. It consists of two main sections: a left panel for input and a right panel for results and explanations.

Left Panel (Input):

- REGULAR EXPRESSION:** `: / gr[ea]y`
- TEST STRING:** `gray` (highlighted in blue) and `grey`
- REGULAR EXPRESSION:** `: / [a-z,A-Z]+`

Right Panel (Results):

- EXPLANATION:** A green button labeled "2 matches (10 steps, 0.0ms)" is present above the results.
- MATCH INFORMATION:** A table showing two matches:

Match	Range	Text
Match 1	0-4	gray
Match 2	5-9	grey

Again, here the class says upper and lower case a-z matches. Some optional ways below are using the wildcard “.” or insert class string in the word as well.

REGULAR EXPRESSION 2 matches (4 steps, 0.0ms)

```
:/ [a-z,A-Z]+ / gm
```

TEST STRING

```
Adviser
Advisor
```

EXPLANATION

MATCH INFORMATION

Match 1	0-7	Adviser
Match 2	8-15	Advisor

REGULAR EXPRESSION 2 matches (16 steps, 0.0ms)

```
:/ Advis.r / gm
```

REGULAR EXPRESSION 2 matches (16 steps, 0.0ms)

```
:/ Advis[oe]r / gm
```

Depending, yes we can use “\D” or “.” followed by “+” and it works, but remember will this work in an actual log? It may. Here, I found using “?” worked well. Remember the “?” says optional and will come in handy.

The screenshot shows a regular expression testing interface with three examples:

- REGULAR EXPRESSION:** `:/ [aesthetic]+` (highlighted in orange) **TEST STRING:** `aesthetic` `esthetic` **EXPLANATION:** 2 matches (4 steps, 0.1ms) **MATCH INFORMATION:**
 - Match 1 0-9 aesthetic
 - Match 2 10-18 esthetic
- REGULAR EXPRESSION:** `:/ a?esthetic` (highlighted in blue) **REGULAR EXPRESSION:** `:/ [ae]?esthetic` (highlighted in orange)

Well that came sooner than expected. Remember “?”? Now is a good time as we say match analog and then what follows optional.

The screenshot shows a regular expression testing interface. On the left, under 'REGULAR EXPRESSION', is the pattern `:/ analog[u]?[e]?`. Under 'TEST STRING' are the words `analog` and `analogue`. A green button at the top right indicates `2 matches (18 steps, 0.0ms)`. On the right, under 'EXPLANATION', is a summary. Under 'MATCH INFORMATION', two matches are listed: Match 1 at index 0-6 for the word `analog`, and Match 2 at index 7-15 for the word `analogue`.

REGULAR EXPRESSION	
<code>:/ analog[u]?[e]?</code>	/ gm

TEST STRING

`analog`
`analogue`

EXPLANATION

MATCH INFORMATION

Match 1 0-6 `analog`

Match 2 7-15 `analogue`

Ok, now this seems work related so lets build a class for hexadecimal. Fairly straight forward but lets keep going.

The screenshot shows a regular expression testing interface with two examples.

Example 1:

- REGULAR EXPRESSION:** `: / [0-9,a-f,A-F]x[0-9,a-f,A-F]{6}`
- TEST STRING:** `0x00000000`, `0xFFFFFFF`, `0x00000000`, `0x00000DF`, `0x000AA00`
- RESULTS:** 5 matches (20 steps, 0.1ms) / gm
- EXPLANATION:** Match 1: 0-8 | `0x00000000`; Match 2: 9-17 | `0xFFFFFFF`; Match 3: 18-26 | `0x00000000`; Match 4: 27-35 | `0x00000DF`; Match 5: 36-44 | `0x000AA00`

Example 2:

- REGULAR EXPRESSION:** `: / [0-9,a-f,A-F]x[0-9,a-f,A-F]+`
- TEST STRING:** (empty)
- RESULTS:** 5 matches (20 steps, 0.1ms) / gm
- EXPLANATION:** Match 1: 0-8 | `0x00000000`; Match 2: 9-17 | `0xFFFFFFF`; Match 3: 18-26 | `0x00000000`; Match 4: 27-35 | `0x00000DF`; Match 5: 36-44 | `0x000AA00`

Lets grab our MAC and see if we can regex that. First hit that CLI and ipconfig /all and go ahead and copy/paste.

```
C:\Users\Chris>ipconfig /all

Connection-specific DNS Suffix . : hsd1.co.comcast.net
Description . . . . . : Intel(R) Wi-Fi 6 AX201 160MHz
Physical Address. . . . . : 44-AF-28-07-E3-01
Link Layer Protocol Type . . . . . : IEEE 802.11
```

Then lets regex it. Again, hexadecimal and some options to grab as a string are the classic “+” or I initially went with “{1,}” which really is the same as “+” in this sense.

The image shows two side-by-side screenshots of a regular expression tester. Both screenshots have a light gray header bar with tabs for "REGULAR EXPRESSION" and "EXPLANATION".

Top Screenshot:

- REGULAR EXPRESSION:** `:/ [0-9,a-f,A-F,-]{1,}`
- TEST STRING:** `44-AF-28-07-E3-01`
- EXPLANATION:** Shows "1 match (2 steps, 0.1ms)" and the match information: Match 1 at index 0-17 with the value `44-AF-28-07-E3-01`.

Bottom Screenshot:

- REGULAR EXPRESSION:** `:/ [0-9,a-f,A-F,-]+`
- TEST STRING:** `44-AF-28-07-E3-01`
- EXPLANATION:** Shows "1 match (2 steps, 0.0ms)" and the match information: Match 1 at index 0-17 with the value `44-AF-28-07-E3-01`.

This time lets just grab the numbers. I'm sure you remember "\d" and its a zipcode so lets tell it 5x using "{5}" and there we go.

The screenshot shows a regular expression testing interface. On the left, under 'REGULAR EXPRESSION', is the pattern `:/ \d{5}` with options `/ gm`. A green bar at the top right indicates `1 match (2 steps, 0.1ms)`. On the left, under 'TEST STRING', is the input `I don't agree that 80210 was a fun show.`. On the right, under 'EXPLANATION' and 'MATCH INFORMATION', the result is shown as Match 1, spanning from index 19 to 24, with the value `80210`.

Individual words? No problem. Character class and away we go.

REGULAR EXPRESSION

27 matches (54 steps, 0.1ms)

/ [A-z] + / gm

TEST STRING

Tim, • Ron • and • Wallace, • knew • from • their • studies • that • regex •
could • be • used • in • looking • for • text • within • an • operating •
systems • and • in • files • in • that • operating

This screenshot shows a regular expression testing interface. The regular expression input field contains ': / [A-z] +' with options for case sensitivity (gm) and a copy button. The test string is a sentence about regex usage. The results section shows 27 matches, each consisting of a match number, its start and end index, and the matched word. The matches are listed vertically, with the first few being 'Tim', 'Ron', 'and', 'Wallace', 'knew', 'from', 'their', 'studies', 'that', 'regex', 'could', 'be', 'used', 'in', 'looking', 'for', 'text', 'within', 'an', 'operating', 'systems', 'and', 'in', 'files', 'in', and 'that'.

MATCH INFORMATION

Match 1	0-3	Tim
Match 2	5-8	Ron
Match 3	9-12	and
Match 4	13-20	Wallace
Match 5	22-26	knew
Match 6	27-31	from
Match 7	32-37	their
Match 8	38-45	studies
Match 9	46-50	that

Can we grab the entire string in one stroke? Of course, but I took the easy way out. What did you come up with?

The screenshot shows a regular expression testing interface. On the left, under 'REGULAR EXPRESSION', is the pattern ': / .+'. To its right, a green button indicates '1 match (2 steps, 0.2ms)'. Below this is a search bar with '/ gm' and a copy icon. Under 'TEST STRING', the input is: 'Christian, •Tyler, •and •Reece, •didn't •like •regex •at •all. •But •they •knew •it •was •important •to •learn •for •lots •of •reasons.' A blue box highlights the entire string. On the right, under 'EXPLANATION', is a collapsed section. Under 'MATCH INFORMATION', it shows 'Match 1 0-115' with a blue underline. To the right, the matched string is displayed: 'Christian, •Tyler, •and •Reece, •didn't •like •regex •at •all. •But •they •knew •it •was •important •to •learn •for •l...'. An upward arrow icon is next to the explanation section.

Again? Ok, this time I also included another way. I said lets catch everything but is this really different than “.+”?

The screenshot shows a regular expression testing interface with two examples.

Example 1: Regular Expression: `/.+`
Test String: "Richard·Simmons·1992·album·is·the·greatest·hip·hop·album·of·all·time,·including·Childish·Gambino."
Match Information: Match 1 [0-97] Richard·Simmons·1992·album·is·the·greatest·hip·hop·album·of·all·time,·including·Childish·Gambino.

Example 2: Regular Expression: `/\[\D,0-9\]+\`
Test String: (empty)
Match Information: No matches found.

I was disappointed in me too. I wanted to try to catch sentences so added a short test sentence at the end. It looks like I did it...but did I? The literal “.” is going to trip up my regex if ever a doctor (Dr.) enters the room. Also, many other reasons this would break, but fun getting to this point. Whats next??

REGULAR EXPRESSION

2 matches (12 steps, 0.0ms)

/ ([A-Z,a-z,0-9,^0-9\s]+.\s?) / gm

TEST STRING

Richard•Simmons•1992•album•is•the•greatest•hip•hop•album•of•
all•time,•including•Childish•Gambino. Adding•for•testing.

EXPLANATION

MATCH INFORMATION

Match	Range	Text
Match 1	0-98	Richard•Simmons•1992 •album•is•the•greatest •hip•hop•album•of•all •time,•including•Childish •Gambino..
Group 1	0-98	Richard•Simmons•1992 •album•is•the•greatest •hip•hop•album•of•all •time,•including•Childish •Gambino..
Match 2	98-117	Adding•for•testing.
Group 1	98-117	Adding•for•testing.

Looks like we have a possible log string now. Lets tackle the date/time. Getting easier? It may not be the most efficient but a quick combination of \d, \s, and literals and we have the dtg selected.

The screenshot shows a regular expression testing tool with the following details:

- REGULAR EXPRESSION:** `:/ \d{2}\/\d{2}\/\d{4}\s\d{2}:\d{2}:\d{2}\.\d{4} / gm`
- TEST STRING:** `10/11/2008 08:15:00.0154 EVID:4140--A 'Extreme failure' has occurred. Your system is been down, your cat has cheezburger!`
- EXPLANATION:** A section showing the breakdown of the regular expression with color-coded matches.
- MATCH INFORMATION:** A table showing the details of the match:

Match 1	0-24	10/11/2008 08:15:00.0154
---------	------	--------------------------

Can we pull out just the EVID number? Doing some reading and testing I found a way to look for EVID and then just provide me what follows. However, regex has multiple ways and play around with it and see what you get.

The screenshot shows a regex testing interface with the following details:

- REGULAR EXPRESSION:** `: / (?=<EVID:)\d{4}`
- TEST STRING:** `10/11/2008•08:15:00.0154•EVID:4140•--A•‘Extreme•failure’•has•occurred.•Your•system•is•been•down,•your•cat•has•cheezburger!`
- EXPLANATION:** The regular expression matches 1 group (41 steps, 0.0ms).
- MATCH INFORMATION:** Match 1 spans from index 30 to 34 and contains the value 4140.

Entire string? Sure, lets snag a \D, \d and away we go.

The screenshot shows a regular expression testing interface. On the left, under 'REGULAR EXPRESSION', the pattern `: / [\D,\d]+` is entered with options `/ gm`. A green button indicates `1 match (2 steps, 0.1ms)`. On the right, under 'EXPLANATION', the 'MATCH INFORMATION' section shows 'Match 1' at index 0-122. The matched text is `10/11/2008·08:15:00.0154·EVID:4140--·A·‘Extreme·failure’·has·occurred.·Your·system·is·been·down,·your·cat·has·cheezburger!`. The entire test string is also visible in the background.

Entire string and the EVID number? Lets group up EVID and we have it completed.

REGULAR EXPRESSION 1 match (290 steps, 0.0ms)

```
// [\D,\d]+((?=EVID:)\d{4})[\D,\d]+
```

/ gm

TEST STRING

```
10/11/2008 08:15:00.0154 EVID:4140 -- A 'Extreme failure' has occurred. Your system is been down, your cat has cheezburger!
```

EXPLANATION

MATCH INFORMATION

Match 1	0-122	10/11/2008 08:15:00.0154 EVID:4140 -- A 'Extreme failure' has occurred. Your system is been down, your cat has cheezburger!
Group 1	30-34	4140

These are some of the basics of regex and how to apply them. There is so much more. Keep on learning.

Bonus: can you match and group the following log section

2024-02-10 00:01:56, User4, open file, success, Document1.docx
2024-02-23 05:40:33, User1, login, success,
2024-02-19 21:09:17, User2, open file, success, Document1.docx
2024-02-12 20:35:58, User3, logout, success,
2024-02-10 19:25:07, User2, login, success,
2024-02-14 08:40:18, User4, login, success,
2024-02-26 07:37:04, User4, delete file, success, Presentation.pptx

Continue for answer

REGULAR EXPRESSION

1 000 matches (42 174 steps, 29.2ms)

```
: / (?<date>\d{4}\-\d{2}\-\d{2})(?  

<time>\s\d{2}\:\d{2}\:\d{2}),\s(?<UserID>User\d),\s(?  

<action>\w{1,}\s?\w{1,})\,\s(?<result>\w+)\,\s?(?<target>.+)?
```

/ gm

TEST STRING

2024-02-27•13:42:25,•User1,•logout,•success,
2024-02-20•06:20:01,•User2,•login,•success,
2024-03-04•02:43:50,•User1,•open•file,•success,•Document1.docx
2024-02-10•21:19:19,•User1,•logout,•success,
2024-02-27•00:24:23,•User2,•open•file,•success,•Report.pdf
2024-02-25•08:48:41,•User1,•login,•success,
2024-02-10•00:22:49,•User1,•password•failure,•failed,
2024-02-09•00:53:03,•User2,•logout,•success,
2024-03-01•12:03:40,•User3,•delete•file,•success,•Presentation.pptx
2024-02-25•11:26:19,•User4,•delete•file,•success,•Report.pdf
2024-02-11•05:32:23,•User2,•edit•file,•success,•Spreadsheet.xlsx
2024-02-19•10:08:08,•User2,•password•failure,•failed,
2024-02-08•13:14:11,•User1,•password•failure,•failed,
2024-02-10•01:49:33,•User4,•password•failure,•failed,
2024-02-24•13:06:58,•User2,•logout,•success

EXPLANATION

MATCH INFORMATION

Match 1	0-65	2024-03-05 •08:44:39, •User3, •edit•file, •success, •Presentation.pptx
Group date	0-10	2024-03-05
Group time	10-19	•08:44:39
Group UserID	21-26	User3
Group action	28-37	edit•file
Group result	39-46	success
Group target	48-65	Presentation.pptx
Match 2	66-120	2024-02-18 •06:40:31, •User2, •password •failure, •failed, •