# 20 Most Common Regular Expressions

Updated on March 5, 2022 by Benjamin



Thousands of regular expressions are written by programmers around the world each day to validate user input or to find common patterns in strings of text. Amidst all this noise, I thought it would be great to create a list of the most common regular expressions for easy access.

So I did some personal research to find out which of those thousands of expressions are the **most commonly used** around the world. Here's my methodology:

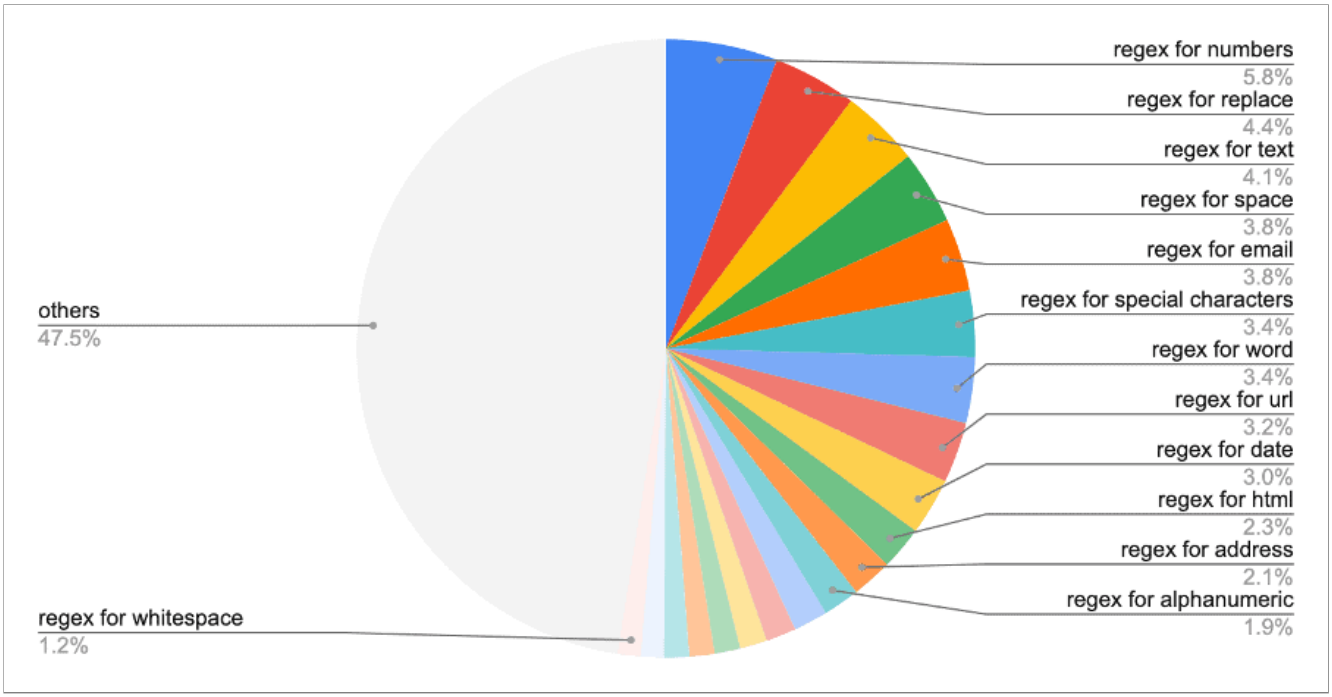
## Finding the Most Common Expressions

My reasoning was that the most common expressions would be those most searched for on Google. So I opened up a new browser window and typed **`regex for a`**... into the search bar, noting down the top 10 AutoComplete suggestions from Google. I did the same for **`regex for b`**..., and continued to the end of the alphabet until I had a list of 260 regular expression search terms.

By the way, this was all done in an incognito window to remove any bias based on my previous searches.

I then entered all of these search terms one by one into Google Trends and compared their average search volumes as measured against the most popular term I could find, which was **`regex for numbers`**. I noted each term's relative search volume down in a spreadsheet and ordered the list in descending order. I then removed some items that seemed similar to others, and voila...

This left me with a list of the **most-searched-for regular expressions**, in order of importance. See the chart below for results.




Most Common Regular Expressions Pie Graph (generated using Google Sheets)

By the way, I don't know of any other place where this data is available on the Internet.

Now, I know that this is not a foolproof method and that numbers in Google Trends vary daily based on changes in real-time search volumes, but this puts us on the right track at least, given that we are working with average searches.

So without further ado, here are the top 20 regular expressions (i.e. regex for ...) from my research:

1. Numbers
2. Replacing a match
3. Text

4. Spaces
5. Email addresses
6. Special characters
7. Words
8. URLs
9. Dates
10. HTML tags
11. Addresses
12. Alphanumeric characters
13. Letters
14. Phone numbers
15. Digits
16. Passwords
17. IP addresses
18. JSON strings
19. Time
20. Whitespace

Now, let's discuss these in detail and come up with a basic expression for each. Click on the links in the list above to go to the relevant sections you are interested in.

## 1. Matching Numbers

Numbers come in all shapes and sizes. Whole numbers, decimal numbers, scientific numbers, etc. Here are a couple of expressions for the most common ones.

To match whole numbers with one or more digits, use the digit character `\d` like this:

```
/\d+/
```

To include negative whole numbers in the list above:

```
/-?\d+/
```

For matching decimal numbers with a period `.` as decimal separator:

```
/-?\d+(\.\d*)?/
```

Replace the `\.` above with a `,` to match a decimal number with a comma separator `,` instead.

## 2. Replacing A Matched Term

One of the most common uses or regex is to replace matched terms with something else. An example would be to find all instances of the word **`colour`** and replace it with the alternate spelling **`color`**. The spelling differs based on where you live in the world. Here's the Javascript code to do just that:

```
var input = "The colour of my shirt matches the colour of my tie.";
var output = input.replace(/colour/, "color");
```

In the second line of this expression, the regular expression to match **`colour`** is written as the first parameter in the "replace()" function, while the second parameter contains the replacement term.

## 3. Matching Text

Text, such as that in a printed book, normally consists of letters (both upper and lowercase) `[A-Za-z]`, numbers `[0-9]`, spaces `\s`, and basic punctuation such as periods commas, dashes, parentheses, etc.:

```
/[A-Za-z0-9\.,;:!?()"'%\-]+/
```

You can adapt the expression above by adding other characters and punctuation marks inside the square brackets as required.

## 4. Matching Spaces

To match one or more spaces, just enter a blank space followed by an appropriate quantifier. Note the space before the one-or-more quantifier `+` below:

```
/ +/
```

Alternatively, you can use the ASCII character code for a space `\x20` for better readability:

```
/\x20+/
```

The whitespace character `\s` cannot be used here since it will also matched other whitespace characters such as tabs and newlines.

## 5. Matching Email Addresses

Email addresses are always tricky since the restrictions on valid email addresses are quite extensive. There is no valid email regex, but here's a simple one that will get you started:

```
/^[A-Za-z0-9!#$%&'"\`”+/\=?^_`{}|~,():;<>[]\-.]*@[A-Za-z0-9-]*\.[A-Za-z]+(?:\.[A-Za-z]+)?(?:\.
[A-Za-z]+)?$/
```

The idea is to start with this expression and build it out to suit your own needs. You can find a list of valid email addresses here and here to test your expression against.

In addition, this handy website provides an entire range of email regular expressions in various languages.

## 6. Matching Special Characters

Matching special characters is easy when you know their ASCII hexadecimal code. You can find codes for various characters from the HEX column in this table (not including the extended ASCII table). For example, matching the tilde sign ~ is matched by:

```
/\x7E/
```

You can also match Unicode characters using the 4-character Unicode hexadecimal code found in the tables here. For instance, the character π is matched by:

```
/\u03A0/
```

You can also match ranges of these characters as follows:

```
/[\x20-\x7E]/
```

```
/[\u03A0-\u03FF]/
```

## 7. Matching Words

Matching words is easily done using the word character `\w`. It matches all uppercase `[A-Z]` and lowercase `[a-z]` letters as well as numbers `[0-9]`:

```
/\w/
```

## 8. Matching URLs

For a general code to match URLs, use the following:

```
/(?:http|https|ftp|mailto|file|data|irc):\/\/[A-Za-z0-9\-]{0,63}(\.[A-Za-z0-9\-]{0,63})+
(:\d{1,4})?\/*(\/*[A-Za-z0-9\-._]+\/*)*(\?.*)?(#.*)?/
```

The expression above has been generated according to the format specified here, and will match **`http`**, **`https`**, **`ftp`**, **`mailto`**, **`file`**, and **`irc`** schemes, and with query strings (preceded by `?`) and fragments (preceded by `#`).

Once again, this expression is hardly foolproof, and is meant as a starting point which you can modify to suit your needs.

## 9. Matching Dates

Dates come in various formats, depending on where you live in the world. Here are a few popular formats you might find useful.

For dates in the format **`yyyy/mm/dd`**:

```
/^\d{4}/(0[1-9]|1[0-2])/(0[1-9]|[12][0-9]|3[01])$/
```

For dates in the format **`mm/dd/yyyy`**:

```
/^(0[1-9]|1[0-2])/(0[1-9]|[12][0-9]|3[01])/\d{4}$/
```

For date in the format **`dd/mm/yyyy`**:

```
/^(0[1-9]|[12][0-9]|3[01])/(0[1-9]|1[0-2])/\d{4}$/
```

You can change the delimiter in the dates above by changing the forward-slash `/` to an alternate delimiter (say the dash symbol `–`).

Take note that these expressions do not guarantee a valid date. For instance, it will accept 31/02/2020 as a valid date even though February only has 28/29 days. It only validates the format of the date.

**ALSO READ:** **Regex for Dates**

## 10. Matching HTML Tags

HTML tags are enclosed in angle brackets **<** and **>**. To match the contents of any word enclosed in angle brackets (including non-standard tags), use the following expression:

```
/<.+>/
```

To match a predefined set of tags (say **<p>**, **<a>**, **<b>**, and **<img>**) along with their corresponding closing tags (but without attributes included), the following expression will do:

```
/<\/?(?:p|a|b|img)(?: \/)?>/
```

And including attributes (e.g. **width="100%"**):

```
/<\/?(?:p|a|b|img)(?: [A-Za-z0-9]+=["'`].+["'`])*(?: \/)?>/
```

## 11. Matching Addresses

The format of physical and postal address differ widely across the world as seen here, and are typically made up of letters (both upper and lower case) **[A-Za-z]**, numbers **[0-9]**, spaces **\s**, various punctuation marks, and more.

These are often written across several lines so a newline character **\n** must also be included with the match:

Here's an expression that fits the bill above:

```
/[A-Za-z0-9\.\s,\-:\n]+/
```

This expressions is by no means exhaustive but can be adapted to suit the needs of the specific country you are targeting.

## 12. Matching Alphanumeric Characters

Matching alphanumeric characters means matching a combination of letters (both upper and lowercase) **[A-Za-z]** and numbers **[0-9]**:

```
/[A-Za-z0-9]/
```

To match both alphanumeric characters and spaces, use the following:

```
/[A-Za-z0-9\s]/
```

**ALSO Read: Regex for Alphanumeric Strings**

## 13. Matching Letters

Matching a specific letter can be done with the simple expression:

```
/a/
```

```
/c/
```

Matching a range of letters:

```
/[aeiou]/
```

Matching all letters, both upper and lowercase:

```
/[A-Za-z]/
```

## 14. Matching Phone numbers

Phone numbers look different depending on which part of the world you are from.

The most popular format is the NANP format, used in the US and various other countries, and which takes the form **`XXX-NXX-XXXX`**. Here's an expression that can match a number of this format:

```
/(?:\(?\d{3})?\)?[- ]?[2-9]\d{2}[- ]?\d{4}/
```

## 15. Matching Digits

Matching digits from 0 to 9 is done with either of the following expressions:

```
/\d/
```

```
/[0-9]/
```

You can also match a digit from a specific list of digits by combining them in square brackets, like this:

```
/[579]/
```

To match more than one digit, use a quantifier. For instance, use the one-or-more quantifier `+` to match one or more digits:

```
/\d+/
```

Or to match a specific number of digits (say 5 digits), use the following expression (you can change the 5 below to the number of digits you require):

```
/\d{5}/
```

## 16. Matching Passwords

Regular expressions shine when testing whether passwords are complex enough, and the best part is that it can be done using Javascript right in the user's browser window.

The following expression will test if a password consists of letters and numbers and is at least 8 characters long:

```
/[A-Za-z0-9]{8,}/
```

To include special characters which are found on most keyboards:

```
/[A-Za-z0-9!?@#$%^&*()\-+\\\/.,:;"'{}\[\]<>~]{8,}/
```

To enforce that certain characters must be present, we use a positive lookahead `(?=...)`. For example, the following will ensure at least one uppercase letter `[A-Z]`, one lowercase letter `[a-z]`, one number `[0-9]`, and one special character, and ensure that the password is at least 8 characters long:

```
/(?=.*[A-Z])(?=.*[a-z])(?=.*\d)(?=.*[!?@#$%^&*()\-+\\\/.,:;"'{}\[\]<>~])[A-Za-z0-9!?@#$%^&*
()\-+\\\/.,:;"'{}\[\]<>~]{8,}/
```

Thanks to this answer for inspiring the expression above.

## 17. Matching IP addresses

IP addresses are used on the Internet to identify components connected to a network. Most computers still use IPv4 addresses, and adoption of the new IPv6 address is still a work in progress.

To match an IPv4 IP address:

```
/(?:25[0-5]|2[0-4]\d|[01]\d{2}|\d{1,2})(?:.(?:25[0-5]|2[0-4]\d|[01]\d{2}|\d{1,2})){3}/
```

To match an IPv6 address:

```
/(?:[A-Fa-f0-9]){0,4}(?: ?:? ?(?:[A-Fa-f0-9]){0,4}){0,7}/
```

IPv6 has various rules for shortening addresses by removing 0s. The expression above is not foolproof and is meant as a starting point that you can adjust to suit your needs, based on the shortening rules you adopt.

## 18. Matching JSON Strings

JSON is a complicated language with nested structures and arrays. A complete validation using regex, however, is possible. Take a look at this answer on Stack Overflow for further information which uses recursion to validate JSON strings.

## 19. Matching Time

To match 12-hour time in the format HH:MM or HH:MM:SS, use these expressions:

```
/(?:12:00|(?:1[01]|[1-9]):(?:[0-5]\d))/
```

```
/(?:12:00:00|(?:1[01]|[1-9]):(?:[0-5]\d):(?:[0-5]\d))/
```

To match 24-hour time in the format HH:MM or HH:MM:SS, use these expression:

```
/(?:2[0-3]|[01]\d|\d):(?:[0-5]\d)/
```

```
/(?:2[0-3]|[01]\d|\d):(?:[0-5]\d):(?:[0-5]\d)/
```

To match 12-hour time with AM/PM, use the following:

```
/^(?:12:00|(?:1[01]|[1-9]):(?:[0-5]\d))\s?((?:A|P)\.?M\.?)$/i
```

**ALSO READ:** **Regex for 12-hour Time with AM/PM**

## 20. Matching Whitespace

To match all whitespace character including a space, tab, carriage return, and newline, simply use the whitespace character `\s`:

```
/\s/
```

Individual whitespace characters include the newline `\n`, carriage return `\r`, tab and `\t` characters:

```
/\n\r\t/
```

If you'd like to match a space only, you can use the ASCII representation of a space `\x20`:

```
/\x20/
```

## Conclusion

The expressions above show the most common uses of regex and provides regular expressions for each one. If you find an error or have a suggestion for improving these, please leave a comment below so we can learn together.