Regular Expressions

Regular Expression Examples

Let Σ = {a, b}.

a = {a}
ab = {ab}
a | b = {a, b}
a^+ = {a, aa, aaa, ... }

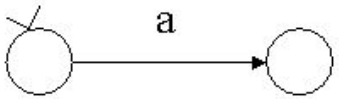
ab* represents the set of strings having a single a followed by zero or more occurrences of b.
That is, it's {a, ab, abb, abbb, ... }

a (b | c) = {ab, ac}

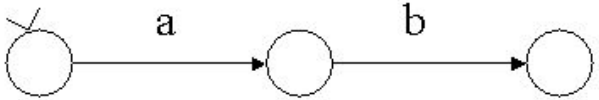
(a | b) (c | d) = {ac, ad, bc, bd}
aa* = a^+ = {a, aa, aaa, ... }

Equivalence of Finite Automata and Regular Expressions

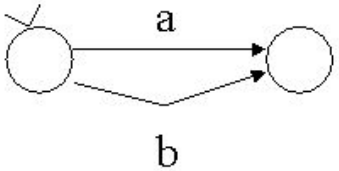
a



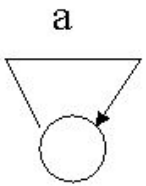
ab

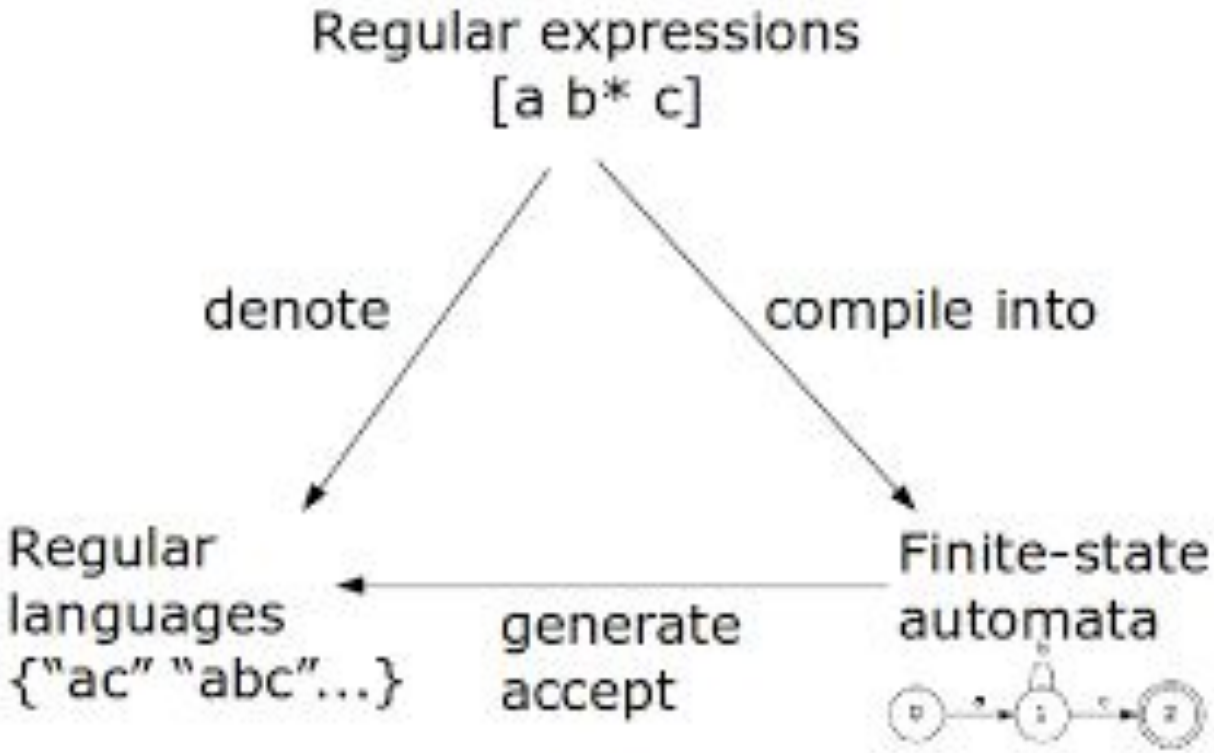


a | b



a*




Regular expressions
[a b* c]
denote
compile into
Regular
languages
{"ac" "abc"...}
generate
accept
Finite-state
automata

Code example - with inline comments

```
/
  (.*)                # any leading prefix
  (flat|shop|unit)    # unit types such as apartment, shop, etc
  \s                  # mandatory space
  (\w+)               # some digits or letters, such as:  10b
  (\s?.*)             # some optional whitespace and then
                      # anything else (suffix)
/x
```

REGEX CHEATSHEET

EVERYTHING YOU NEED TO KNOW WHEN WORKING WITH REGULAR EXPRESSIONS (POSIX STANDARDS)

Repetitions of a pattern

*	at least 0 times	**abc*** "ab" followed by zero or more "c"
+	at least 1 time	**abc+** "ab" followed by one or more "c"
?	at most 1 time	**abc?** "ab" followed by zero or one "c"
{n}	exactly n times	**abc{2}** "ab" followed by 2 "c"
{n,}	at least n times	**abc{3,}** "ab" followed by at least 3 "c"
{n, m}	at least n times, at most m times	**abc{3,5}** "ab" followed by 3 up to 5 "c"

Positions of a pattern

^	the start of the string	**^starts** string starting with "starts"
$	the end of the string	**ends$** string ending with "ends"
\b	empty string at edge of a word	**\\bab** matches "ab" in "abc" but not in "dabc"
\B	empty string not at edge of a word	**\\Bab** matches "ab" in "dabc" but not in "abc"

1. Matching exact characters:
   - Regex: `hello`
   - Matches: Any string containing "hello" [1]
2. Matching alternative characters:
   - Regex: `gr[ae]y`
   - Matches: "gray" or "grey" [1]
3. Optional characters:
   - Regex: `colou?r`
   - Matches: "color" or "colour" [1]
4. Repetition:
   - Regex: `go+gle`
   - Matches: "gogle", "google", "gooogle", and so on [1]
5. Matching digits:
   - Regex: `\d+` or `[0-9]+`
   - Matches: Any sequence of one or more digits [2]

6. Matching word characters:
   - Regex: `\w+`
   - Matches: Any sequence of one or more letters, digits, or underscores [2]
7. Matching the start of a string:
   - Regex: `^dog`
   - Matches: Any string that begins with "dog" [1]
8. Matching the end of a string:
   - Regex: `dog$`
   - Matches: Any string that ends with "dog" [1]
9. Matching a specific number of occurrences:
   - Regex: `z{3,6}`
   - Matches: "zzz", "zzzz", "zzzzz", or "zzzzzz" [1]
10. Matching identifiers:
    - Regex: `[a-zA-Z_][0-9a-zA-Z_]*`
    - Matches: Strings that start with a letter or underscore, followed by any number of letters, digits, or underscores [2]

regexr.com

Relaunch to update

DeployGPT | Amazon Web Servi... | All Bookmarks

Untitled Pattern | Save (cmd-s) | New | by gskinner | GitHub | Sign In

Menu

- Pattern Settings
- My Patterns
- Cheatsheet
- RegEx Reference
- Community Patterns
- Help

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

Expression | JavaScript | Flags

```
/[0-9]{4}-[0-9]{4}/g
```

Text | **Tests** NEW | **4 matches** (0.2ms)

```
Jane flipped through the files:

A Visa charge on 4111-1111-1111-1111 for a luxury watch.
A MasterCard purchase using 5500 0000 0000 0004 at an electronics store.
An American Express transaction from 3400-000000-00009 for first-class airline tickets.
And an attempt to buy a vintage car with 6011 0000 0000 0004 (Discover).
The kicker? Each charge came with a different name and email address.

"John Doe, email: john.doe@example.com," she read aloud. "Then we have a 'Mike Rogers' with sales-
dept@company.co.uk and a 'Lisa Tran' using admin123@business.net."

"These don't seem like ordinary buyers," the bank rep murmured.

Jane agreed. "And check this out—some of the contact numbers are weird: (123) 456-7890, +1-800-555-0199, even a
UK number: +44 20 7946 0958."
```

Must include @

Must include dot

jovialsmith92@example.com

The id should be to the left of @ and may contain alphabets, digits and special characters.

The domain name should be between the @ and the dot. It may include alphabets, digits and - only.

The domain extension should come after the dot and may include alphabets o

Regular Expression E-mail Matching Example

Regular expression boundary

Match anything contained within brackets

Match as many times as possible

Match the @ symbol

Match upper and lower case A through Z

```
/[\w._%+-]+@[\w.-]+\.[a-zA-Z]{2,4}/
```

Match ., _, %, +, and - if found

Match a single period

Match at least two times but no more than four times

Match any character A-Z upper or lower case and any number 0 to 9

Python Example

Python's `re` module allows regex matching easily.

python Copy Edit

```python
import re

email_regex = r"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}"
text = "Contact us at support@example.com or sales@example.org."

matches = re.findall(email_regex, text)
print("Emails found:", matches)
```

Output:

nginx Copy Edit

```nginx
Emails found: ['support@example.com', 'sales@example.org']
```

JavaScript Example

JavaScript's `RegExp` and `.match()` method can extract emails.

javascript Copy Edit

```javascript
const emailRegex = /[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}/g;
const text = "Contact us at support@example.com or sales@example.org.";

const matches = text.match(emailRegex);
console.log("Emails found:", matches);
```

Output:

nginx Copy Edit

```nginx
Emails found: ['support@example.com', 'sales@example.org']
```

Go Example

Go has a built-in `regexp` package for regex matching.

go Copy Edit

```go
package main

import (
    "fmt"
    "regexp"
)

func main() {
    emailRegex := `[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}`
    text := "Contact us at support@example.com or sales@example.org."

    re := regexp.MustCompile(emailRegex)
    matches := re.FindAllString(text, -1)

    fmt.Println("Emails found:", matches)
}
```

Output:

nginx Copy Edit

```nginx
Emails found: [support@example.com sales@example.org]
```

Java Example for Extracting Emails Using Regex

java

Copy Edit

```java
import java.util.regex.*;
import java.util.*;

public class EmailRegexExample {
    public static void main(String[] args) {
        String text = "Contact us at support@example.com or sales@example.org.";
        String emailRegex = "[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}";

        Pattern pattern = Pattern.compile(emailRegex);
        Matcher matcher = pattern.matcher(text);

        List<String> emails = new ArrayList<>();
        while (matcher.find()) {
            emails.add(matcher.group());
        }

        System.out.println("Emails found: " + emails);
    }
}
```

https://regexr.com/4tmui

https://regexr.com/3e48o

Expression JavaScript ▾ Flags ▾

```
/\b(\w)[ \t,'"]*(?:(\w)[ \t,'"]*(?:(\w)[ \t,'"]*(?:(\w)[ \t,'"]*(?:(\w)[ \t,'"]*(?:(\w)[ \t,'"]*(?:(\w)[ \t,'"]*(?:(\w)[ \t,'"]*(?
```

Text **Tests** NEW — **31 matches** (0.0ms)

```
abcdcba
abcddcba

abcdedcba
abcdeedcba
abcdefedcba
abcdeffedcba
abcdefgfedcba
abcdefggfedcba
abcdefghijjihgfedcba
abc defg hijjihgfedcba
abcdefghijji  hg fe dc b a
a b c d e f g   h   i j   j  i  h g f e dc  b a

AmanaplanacanalPanama
A man a plan a canal Panama
```

Tools Replace List Details Explain

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

`\b` **Word boundary.** Matches a word boundary position between a word character and non-word character or position (start / end of string).

`(` **Capturing group #1.** Groups multiple tokens together and creates a capture group for extracting a substring or using a backreference.

- `\w` **Word.** Matches any word character (alphanumeric & underscore).

`)`

`[` **Character set.** Match any character in the set.

- **Character.** Matches a SPACE character (char code 32).
- `\t` **Escaped character.** Matches a TAB character (char code 9).