Module 10
Data Processing and Text
Internet Users in the world: 3,587,520,067
Total number of Websites: 1,162,381,089
Emails sent today: 82,167,078,187

Google searches today: 1,819,868,567
Blog posts written today: 1,699,641
Tweets sent today: 231,740,961

Videos viewed today on YouTube: 2,104,290,745
Photos uploaded today on Instagram: 23,682,137
Tumblr posts today: 37,451,716
TRAFFIC FROM MOBILE & ONLINE MESSAGING TO REACH 438 BILLION PER DAY BY 2019

Hampshire, UK: 6th July 2015: New data from Juniper Research has shown that mobile and online messaging traffic will reach 160 trillion per annum by 2019, up from 94.2 trillion this year – equating to approximately 438 billion messages sent and received by users on a daily basis by 2019. These figures incorporate SMS, MMS, IM (Instant Messaging), Social Media and Email.

Last year, email accounted for the largest share of traffic, at around 35 trillion messages per year – although almost 80% of this figure (28 trillion) can be categorised as spam. However, within the next 12 months IM will overtake email generating almost 43 trillion messages annually.
<table>
<thead>
<tr>
<th><strong>YouTube Company Statistics</strong></th>
<th><strong>Data</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of people who use YouTube</td>
<td>1,325,000,000</td>
</tr>
<tr>
<td>Hours of video uploaded to YouTube every minute</td>
<td>300 hours</td>
</tr>
<tr>
<td>Number of videos viewed on YouTube everyday</td>
<td>4,950,000,000</td>
</tr>
<tr>
<td>Number of unique visits to YouTube every month</td>
<td>900,000,000</td>
</tr>
<tr>
<td>Total number of hours of video watched on YouTube each month</td>
<td>3.25 billion hours</td>
</tr>
<tr>
<td>Number of YouTube videos that have generated over 1 billion views</td>
<td>10,113</td>
</tr>
<tr>
<td>Percent of YouTube visitors that come from outside the U.S.</td>
<td>70 %</td>
</tr>
<tr>
<td>Number of countries with localized versions of YouTube</td>
<td>42</td>
</tr>
<tr>
<td>Total number of languages Youtube is broadcast in</td>
<td>54</td>
</tr>
<tr>
<td>User submitted video with the most views – “Charlie bit my finger”</td>
<td>829,000,000</td>
</tr>
<tr>
<td>Average number of mobile YouTube video views per day</td>
<td>1,000,000,000</td>
</tr>
<tr>
<td>Average time spent on YouTube per mobile session</td>
<td>40 minutes</td>
</tr>
<tr>
<td>Average YouTube partner channel payout per 5,000 views</td>
<td>$0.32</td>
</tr>
<tr>
<td><strong>YouTube Company Statistics</strong></td>
<td><strong>Data</strong></td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Total number of people who use YouTube</td>
<td>1,325,000,000</td>
</tr>
<tr>
<td>Hours of video uploaded to YouTube every minute</td>
<td>300 hours</td>
</tr>
<tr>
<td>Number of videos viewed on YouTube everyday</td>
<td>4,950,000,000</td>
</tr>
<tr>
<td>Number of unique visits to YouTube every month</td>
<td>900,000,000</td>
</tr>
<tr>
<td>Total number of hours of video watched on YouTube each month</td>
<td>3.25 billion hours</td>
</tr>
<tr>
<td>Number of YouTube videos that have generated over 1 billion views</td>
<td>10,113</td>
</tr>
<tr>
<td>Percent of YouTube visitors that come from outside the U.S.</td>
<td>70 %</td>
</tr>
<tr>
<td>Number of countries with localized versions of YouTube</td>
<td>42</td>
</tr>
<tr>
<td>Total number of languages Youtube is broadcast in</td>
<td>54</td>
</tr>
<tr>
<td>User submitted video with the most views - “Charlie bit my finger”</td>
<td>829,000,000</td>
</tr>
<tr>
<td>Average number of mobile YouTube video views per day</td>
<td>1,000,000,000</td>
</tr>
<tr>
<td>Average time spent on YouTube per mobile session</td>
<td>40 minutes</td>
</tr>
<tr>
<td>Average YouTube partner channel payout per 5,000 views</td>
<td>$0.32</td>
</tr>
</tbody>
</table>
Netflix and YouTube Are America's Biggest Traffic Hogs

Share of peak period downstream traffic in North America, by application*

Netflix: 31.62%
YouTube: 18.62%
HTTP: 9.74%
BitTorrent: 4.05%
iTunes: 3.27%
Other MPEG: 2.60%
SSL: 2.05%
Amazon Prime Video: 1.61%
Facebook: 1.31%
Hulu: 1.29%
Other: 23.77%

* September 2013. Fixed access only.
Data challenges

• Creating it
• Storing it
• Moving it around
• Keeping it private
Data challenges

• Creating it
• Storing it
• Moving it around
• Keeping it private

• Making sense of it
Searching, indexing
Collecting, Correlating, Recommending
Collecting, Correlating, Recommending
Online Advertising - Convert Shoppers with Relevant Ads
www.criteo.com/Mobile
Make More Sales with Criteo Today.
10,000 brands · 130 countries
Services: Transparent CPC Pricing, Unparalleled Technology, Dynamic Creative, Cross ...

Contact Us
Tell Us A Little About Yourself.
Find a Local Office Near You.

What We Do
Driving Better Marketing Results.
Learn About Our Technology Today

Advertising Online - Marketing360.com
www.marketing360.com/OnlineAdvertising  +1 855-773-8169
#1 Marketing Platform® For Advertising Online. Free Demo!

Online Advertising - Reach More Customers Online
www.rogersoutrank.com/Client_Leads
Demo OutRank's Powerful Platform.

Online advertising - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/Online_advertising
Online advertising, also called online marketing or Internet advertising or web advertising, is a form of marketing and advertising which uses the Internet to ... Display advertising - Web banner - Mobile advertising - Paywall
Rose repeats to Hammerbacher—who’s a founder of data analytics company Cloudera—a line from an interview he gave Businessweek back when he was an early employee hustling stats for Harvard bud Zuckerberg at Facebook:

“The best minds of my generation are thinking about how to make people click ads.”

And Rose, in his politeness, left off the last part of the line: “That sucks.”
# Patterns, trends, predictions

## Quarantine

<table>
<thead>
<tr>
<th>My Folders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Priority Mail -</td>
</tr>
<tr>
<td>Quarantined (0)</td>
</tr>
<tr>
<td>Low Priority Mail - Delivered</td>
</tr>
<tr>
<td>Spam - Quarantined (60)</td>
</tr>
</tbody>
</table>

## Spam - Quarantined

<table>
<thead>
<tr>
<th>Score</th>
<th>From</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td><a href="mailto:vanzari@aramisfeeling.ro">vanzari@aramisfeeling.ro</a></td>
<td>Cadoul tau e la un cl</td>
</tr>
<tr>
<td>99</td>
<td><a href="mailto:Aden@phwwemoty.org">Aden@phwwemoty.org</a></td>
<td><em><strong><strong>SPAM</strong></strong></em>* To l</td>
</tr>
<tr>
<td>99</td>
<td><a href="mailto:rzljqzoaeqywpx@cgl.uwaterloo.ca">rzljqzoaeqywpx@cgl.uwaterloo.ca</a></td>
<td><em><strong><strong>SPAM</strong></strong></em>*</td>
</tr>
<tr>
<td>99</td>
<td><a href="mailto:fymvsmqxcof@cgl.uwaterloo.ca">fymvsmqxcof@cgl.uwaterloo.ca</a></td>
<td><em><strong><strong>SPAM</strong></strong></em>*</td>
</tr>
<tr>
<td>99</td>
<td><a href="mailto:mwhnskic@cgl.uwaterloo.ca">mwhnskic@cgl.uwaterloo.ca</a></td>
<td><em><strong><strong>SPAM</strong></strong></em>*</td>
</tr>
<tr>
<td>99</td>
<td><a href="mailto:amy-manges@goodsnfits.com">amy-manges@goodsnfits.com</a></td>
<td><em><strong><strong>SPAM</strong></strong></em>* See</td>
</tr>
<tr>
<td>99</td>
<td><a href="mailto:vcxtuprei@cgl.uwaterloo.ca">vcxtuprei@cgl.uwaterloo.ca</a></td>
<td><em><strong><strong>SPAM</strong></strong></em>*</td>
</tr>
<tr>
<td>99</td>
<td><a href="mailto:syxlnxsg@cgl.uwaterloo.ca">syxlnxsg@cgl.uwaterloo.ca</a></td>
<td><em><strong><strong>SPAM</strong></strong></em>*</td>
</tr>
<tr>
<td>99</td>
<td><a href="mailto:vlvorxrcdk@cgl.uwaterloo.ca">vlvorxrcdk@cgl.uwaterloo.ca</a></td>
<td><em><strong><strong>SPAM</strong></strong></em>*</td>
</tr>
<tr>
<td>99</td>
<td><a href="mailto:rjtirt@cgl.uwaterloo.ca">rjtirt@cgl.uwaterloo.ca</a></td>
<td><em><strong><strong>SPAM</strong></strong></em>*</td>
</tr>
<tr>
<td>99</td>
<td><a href="mailto:sjwsyf@cgl.uwaterloo.ca">sjwsyf@cgl.uwaterloo.ca</a></td>
<td><em><strong><strong>SPAM</strong></strong></em>*</td>
</tr>
<tr>
<td>99</td>
<td><a href="mailto:nxrvghoxqbewg@cgl.uwaterloo.ca">nxrvghoxqbewg@cgl.uwaterloo.ca</a></td>
<td><em><strong><strong>SPAM</strong></strong></em>*</td>
</tr>
<tr>
<td>99</td>
<td><a href="mailto:sdsjocuiv@cgl.uwaterloo.ca">sdsjocuiv@cgl.uwaterloo.ca</a></td>
<td><em><strong><strong>SPAM</strong></strong></em>*</td>
</tr>
<tr>
<td>99</td>
<td><a href="mailto:laryfhakexa@cgl.uwaterloo.ca">laryfhakexa@cgl.uwaterloo.ca</a></td>
<td><em><strong><strong>SPAM</strong></strong></em>*</td>
</tr>
<tr>
<td>99</td>
<td><a href="mailto:wcjhpjopvxk@cgl.uwaterloo.ca">wcjhpjopvxk@cgl.uwaterloo.ca</a></td>
<td><em><strong><strong>SPAM</strong></strong></em>*</td>
</tr>
<tr>
<td>99</td>
<td><a href="mailto:mmprig@cgl.uwaterloo.ca">mmprig@cgl.uwaterloo.ca</a></td>
<td><em><strong><strong>SPAM</strong></strong></em>*</td>
</tr>
<tr>
<td>99</td>
<td><a href="mailto:wtlby@cgl.uwaterloo.ca">wtlby@cgl.uwaterloo.ca</a></td>
<td><em><strong><strong>SPAM</strong></strong></em>*</td>
</tr>
</tbody>
</table>
Welcome To MalaysianCupid.com!

Dear Member

Congratulations on joining MalaysianCupid.com! This email contains your logon information and important information that will help you get the most out of your membership and help you to find your perfect partner.

YOUR LOGIN DETAILS:
Login Email: azriel@gmail.com
Your Password is: ayang78

Please save this email for future reference!
Thanks for your payment, processed on June 02, 2015.

Hello, AZRIEL TEPPER

American Express Card
81009

We processed your scheduled payment.

$987.91

PROCESSED ON
June 02, 2015

It's processed today - but give us 24 - 36 hours for your payment to appear online.

View your account.

Thanks for your Card Membership,
American Express Customer Care

Was this e-mail helpful? Give us your feedback

STAY CONNECTED

@AskAmex
Amex Customer Care, at your service.

Community_@Amex
Your questions. Your interests. Your community.

Amex Mobile available on the App Store℠ or Google Play™
1700 Years of Global Temperature Change from Proxy Data

Temperature Change (°F)

- Uncertainty
- Proxy-based records
- Thermometer-based records

Year

300 500 700 900 1100 1300 1500 1700 1900

Medieval Warm Period
Little Ice Age
The Data That Turned the World Upside Down

HANNES GRASSEGGER AND MIKAEL KROGERUS
Jan 28  2017, 9:15am

Psychologist Michal Kosinski developed a method to analyze people in minute detail based on their Facebook activity. Did a similar tool help propel Donald Trump to victory? Two reporters from Zurich-based Das Magazin went data-gathering.

An earlier version of this story appeared in Das Magazin in December.
Vault 7: CIA Hacking Tools Revealed

Contents

- Press Release
- Analysis
- Examples
The shape of data

How is your information organized? How do the parts relate to each other?

These questions profoundly affect the tools you use and the code you write.
Call me Ishmael. Some years ago—never mind how long precisely—having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself involuntarily pausing before coffin warehouses, and bringing up the rear of every funeral I meet; and especially whenever my hypos get such an upper hand of me, that it requires a strong moral principle to prevent me from deliberately stepping into the street, and methodically knocking people's hats off—then, I account it high time to get to sea as soon as I can. This is my substitute for pistol and ball. With a philosophical flourish Cato throws himself upon his sword; I quietly take to the ship. There is nothing surprising in this. If they but knew it, almost all men in their degree, some time
TRUMP: Chief Justice Roberts, President Carter, President Clinton, President Bush, President Obama, fellow Americans and people of the world, thank you.

(APPLAUSE) We, the citizens of America, are now joined in a great national effort to rebuild our country and restore its promise for all of our people.

(APPLAUSE)

Together, we will determine the course of America and the world for many, many years to come. We will face challenges, we will confront hardships, but we will get the job done.

Every four years, we gather on these steps to carry out the orderly and peaceful transfer of power, and we are grateful to President Obama and First Lady Michelle Obama for their gracious aid throughout this transition. They have been magnificent. Thank you.

(APPLAUSE)
McCarthy  Faulkner

medium.com/@neuroecology/punctuation-in-novels-8f316d542ec4
Hi Craig,

---

Rishabh Moudgil
Internet Message Format

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2001). All Rights Reserved.

Abstract

This standard specifies a syntax for text messages that are sent between computer users, within the framework of "electronic mail" messages. This standard supersedes the one specified in Request For Comments (RFC) 822, "Standard for the Format of ARPA Internet Text Messages", updating it to reflect current practice and incorporating incremental changes that were specified in other RFCs.
Sequence

46.12 47.88 46.32 45.27 44.32 43.87 44.23 42.95 41.74 40.69
41.68 40.73 40.75 40.55 39.39 39.27 40.89 41.22 40.57
40.43 40.58 39.93 41.08 40.00 37.64 37.46 37.16 36.76 35.65
36.31 37.32 35.55 34.98 34.72 34.55 36.12 36.76 37.62 36.36
37.88 36.59 37.13

The Right Honourable Justin Trudeau
The Right Honourable Stephen Harper
The Right Honourable Paul Edgar Philippe Martin
The Right Honourable Joseph Jacques Jean Chrétien
The Right Honourable A. Kim Campbell
The Right Honourable Martin Brian Mulroney
The Right Honourable John Napier Turner
The Right Honourable Pierre Elliott Trudeau
The Right Honourable Charles Joseph Clark
The Right Honourable Pierre Elliott Trudeau
The Right Honourable Lester Bowles Pearson
The Right Honourable John George Diefenbaker
The Right Honourable Louis Stephen St-Laurent
The Right Honourable William Lyon Mackenzie King
The Right Honourable Richard Bedford Bennett
The Right Honourable William Lyon Mackenzie King
The Right Honourable Arthur Meighen
The Right Honourable William Lyon Mackenzie King
## Dictionary

Associate a set of *keys* with a set of *values*. Ask for the value associated with any key without examining every other key/value pair.

<table>
<thead>
<tr>
<th>Year</th>
<th>City, Country</th>
<th>Year</th>
<th>City, Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1896</td>
<td>Athens, Greece</td>
<td>1968</td>
<td>Mexico City, Mexico</td>
</tr>
<tr>
<td>1900</td>
<td>Paris, France</td>
<td>1972</td>
<td>Munich, West Germany</td>
</tr>
<tr>
<td>1904</td>
<td>St. Louis, United States</td>
<td>1976</td>
<td>Montréal, Canada</td>
</tr>
<tr>
<td>1908</td>
<td>London, United Kingdom</td>
<td>1980</td>
<td>Moscow, Soviet Union</td>
</tr>
<tr>
<td>1912</td>
<td>Stockholm, Sweden</td>
<td>1984</td>
<td>Los Angeles, United States</td>
</tr>
<tr>
<td>1920</td>
<td>Antwerp, Belgium</td>
<td>1988</td>
<td>Seoul, South Korea</td>
</tr>
<tr>
<td>1924</td>
<td>Paris, France</td>
<td>1992</td>
<td>Barcelona, Spain</td>
</tr>
<tr>
<td>1928</td>
<td>Amsterdam, Netherlands</td>
<td>1996</td>
<td>Atlanta, United States</td>
</tr>
<tr>
<td>1932</td>
<td>Los Angeles, United States</td>
<td>2000</td>
<td>Sydney, Australia</td>
</tr>
<tr>
<td>1936</td>
<td>Berlin, Germany</td>
<td>2004</td>
<td>Athens, Greece</td>
</tr>
<tr>
<td>1948</td>
<td>London, United Kingdom</td>
<td>2008</td>
<td>Beijing, China</td>
</tr>
<tr>
<td>1952</td>
<td>Helsinki, Finland</td>
<td>2012</td>
<td>London, United Kingdom</td>
</tr>
<tr>
<td>1956</td>
<td>Melbourne, Australia</td>
<td>2016</td>
<td>Rio de Janeiro, Brazil</td>
</tr>
<tr>
<td>1960</td>
<td>Rome, Italy</td>
<td>2020</td>
<td>Tokyo, Japan</td>
</tr>
<tr>
<td>1964</td>
<td>Tokyo, Japan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SONG</td>
<td>ARTIST</td>
<td>ALBUM</td>
<td>Updated</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------</td>
<td>---------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Ways To Go - Margot Mix</td>
<td>Weval, Margot</td>
<td>Weval Remix</td>
<td>11 hours ago</td>
</tr>
<tr>
<td>Death Is A Girl</td>
<td>Mini Mansions</td>
<td>The Great Pretender</td>
<td>11 hours ago</td>
</tr>
<tr>
<td>Jumbo</td>
<td>Underworld</td>
<td>Beaucoup Fish</td>
<td>11 hours ago</td>
</tr>
<tr>
<td>Bug Powder Dust</td>
<td>The Mysterons</td>
<td>Meandering</td>
<td>11 hours ago</td>
</tr>
<tr>
<td>...To Have No Answer</td>
<td>Flock of Dimes</td>
<td>If You See Me, Sa...</td>
<td>11 hours ago</td>
</tr>
<tr>
<td>I'll Cut You Down</td>
<td>Uncle Acid &amp; The...</td>
<td>Blood Lust</td>
<td>11 hours ago</td>
</tr>
<tr>
<td>L'enfer ce n'est pas les autres c'est moi</td>
<td>The Eye Of Time</td>
<td>Myth I: A Last Da...</td>
<td>11 hours ago</td>
</tr>
<tr>
<td>Terrain</td>
<td>pg.lost</td>
<td>Key</td>
<td>11 hours ago</td>
</tr>
</tbody>
</table>
String operations

String wd = "...";
int len = wd.length();
char c = wd.charAt(2);
String str3 = str1 + str2;
if( str1.equals( str2 ) ) { ... }
String[] words = splitTokens( str1 );
String operations

String wd = "...";
int len = wd.length();
char c = wd.charAt(2);
String str3 = str1 + str2;
if( str1.equals( str2 ) ) { ... }
String[] words = splitTokens( str1 );

Initialize a variable from a string literal
String operations

```java
String wd = "...";
int len = wd.length();
char c = wd.charAt(2);
String str3 = str1 + str2;
if( str1.equals( str2 ) ) { ... }
String[] words = splitTokens( str1 );
```

Count the number of characters in a string
String operations

String wd = "...";
int len = wd.length();
char c = wd.charAt(2);
String str3 = str1 + str2;
if( str1.equals( str2 ) ) { ... }
String[] words = splitTokens( str1 );

Extract a character from a string. Like accessing an array.
String operations

String wd = "...";
int len = wd.length();
char c = wd.charAt(2);
String str3 = str1 + str2;
if (str1.equals(str2)) {
    ...
} 
String[] words = splitTokens(str1);
String operations

String wd = "...";
int len = wd.length();
char c = wd.charAt(2);
String str3 = str1 + str2;
if( str1.equals( str2 ) ) { ... }
String[] words = splitTokens( str1 );
String operations

String wd = "...";
int len = wd.length();
char c = wd.charAt(2);
String str3 = str1 + str2;
if( str1.equals( str2 ) ) { ... }
String[] words = splitTokens( str1 );

Break a string into words by looking for whitespace
Messier text

String[] splitTokens(String text, String delims) {
  ...
}

Break the long string `text` into “words”, where the characters in `delims` (and not whitespace) are treated as breakpoints.

String trim(String text) {
  ...
}

Return a copy of `text` with any excess whitespace removed from the start and end.
Example: the Region of Waterloo’s list of reserved street names

<table>
<thead>
<tr>
<th>FullStreetName</th>
<th>Municipality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abbey Glen</td>
<td>Kitchener</td>
</tr>
<tr>
<td>Aberle</td>
<td>Woolwich</td>
</tr>
<tr>
<td>Abeth</td>
<td>Kitchener</td>
</tr>
<tr>
<td>Abitibi</td>
<td>Cambridge</td>
</tr>
<tr>
<td>Able</td>
<td>Cambridge</td>
</tr>
<tr>
<td>Abram Clemens St</td>
<td>Kitchener</td>
</tr>
<tr>
<td>Accobee</td>
<td>Cambridge</td>
</tr>
<tr>
<td>Adair</td>
<td>Cambridge</td>
</tr>
<tr>
<td>Adcock</td>
<td>Region of Waterloo</td>
</tr>
<tr>
<td>Addlev</td>
<td>Cambridge</td>
</tr>
</tbody>
</table>

Reading the dictionary

A
a
aa
aal
aalii
aam
Aani
aardvark
aardwolf
Aaron
Aaronic
Aaronical
Aaronite
Aaronitic
Aaru
Ab
aba
Ababdeh
Ababua
abac
abaca
abacate
abacay
abacinate
abacination
abaciscus
Reading the dictionary

Find the longest word
Find all words with three or more Ys
Find all words ending with MT
Find all words starting with TM
Find all words ending with DOUS
Find all words containing UFA
Find all words ending in GRY
Find all palindromes
Find words with three consecutive double letters
Find the longest word whose letters are in alphabetical order
Find the longest word with no
Finding things in strings

```java
if( str.contains( "abc" ) ) { ... }

Check if the string `str` has the substring “abc” anywhere inside of it.

```java
if( str.startsWith( "def" ) ) { ... }
if( str.endsWith( "ghi" ) ) { ... }
```

Look for a substring specifically at the start or end of a string.
Writing a spellchecker

With the dictionary at our disposal, it’s easy to check if a given string is a word.

```java
String[] dict;

void setup() {
    dict = loadStrings( "words.txt" );
}

boolean isWord( String word ) {
}
```
Writing a spellchecker

With the dictionary at our disposal, it’s easy to check if a given string is a word.

```java
String[] dict;

void setup() {
    dict = loadStrings( "words.txt" );
}

boolean isWord( String word ) {
    for ( int idx = 0; idx < dict.length; ++idx ) {
        if ( dict[idx].equals( word ) ) {
            return true;
        }
    }
    return false;
}
```
boolean isWord( String word ) {
    for ( int idx = 0; idx < dict.length; ++idx ) {
        if ( dict[idx].equals( word ) ) {
            return true;
        }
    }
    return false;
}

Looping over every word works, but it’s painfully slow, especially when the word isn’t there!
Dictionaries

In programming, a *dictionary* is a mapping from a set of *keys* to a set of *values*. Any given key may have at most one associated value.

- Year → Olympic host city
- Name → Phone number
- Student ID number → Exam seating code
- Clicker ID → Student ID number
- Server name → IP address
Dictionaries

Dictionary operations we might care about:

• Look up the value associated with a given key
• Ask if the dictionary has a given key
• Add a new key to the dictionary, with its associated value
• Remove a key and its value from the dictionary
Processing includes a few handy dictionary classes, where the keys are Strings:

- IntDict: map Strings to ints
- FloatDict: map Strings to floats
- StringDict: map Strings to Strings
Processing includes a few handy dictionary classes, where the keys are Strings:

- **IntDict**: map Strings to ints
- **FloatDict**: map Strings to floats
- **StringDict**: map Strings to Strings

Java allows more-or-less arbitrary mappings between keys and values:

- `java.util.TreeMap<K,V>`: map any key type `K` to any value type `V`. 
**IntDict** myDict = new IntDict();

Create a new, empty dictionary
IntDict myDict = new IntDict();

Create a new, empty dictionary

myDict.set("Kumquat", 13);
myDict.set("Durian", 19);

Add a new key to the dictionary, with its associated value
```java
IntDict myDict = new IntDict;

Create a new, empty dictionary

myDict.set( "Kumquat", 13 );
myDict.set( "Durian", 19 );

Add a new key to the dictionary, with its associated value

println( myDict.get( "Kumquat" ) );

Look up the value associated with a given key
```
IntDict myDict = new IntDict();

Create a new, empty dictionary

myDict.set( "Kumquat", 13 );
myDict.set( "Durian", 19 );

Add a new key to the dictionary, with its associated value

println( myDict.get( "Kumquat" ) );

Look up the value associated with a given key

if( myDict.hasKey( "Rambutan" ) ) { ... }

Ask if the dictionary has a given key
IntDict myDict = new IntDict();
    Create a new, empty dictionary

myDict.set( "Kumquat", 13 );
myDict.set( "Durian", 19 );
    Add a new key to the dictionary, with its associated value

println( myDict.get( "Kumquat" ) );
    Look up the value associated with a given key

if( myDict.hasKey( "Rambutan" ) ) { ... }
    Ask if the dictionary has a given key

myDict.remove( "Durian" );
    Remove a key and its value from the dictionary
Writing a spellchecker

String[] dict;

void setup() {
    dict = loadStrings( "words.txt" );
}

boolean isWord( String word ) {
    for ( int idx = 0; idx < dict.length; ++idx ) {
        if ( dict[idx].equals( word ) ) {
            return true;
        }
    }
    return false;
}
Writing a spellchecker

```java
IntDict myDict;

void setup()
{
    String[] words = loadStrings("words.txt");
    for( int idx = 0; idx < words.length; ++idx ) {
        myDict.set( words[idx], 1 );
    }
}

boolean isWord( String word )
{
    return myDict.hasKey( word );
}
```
Writing a spellchecker

```java
IntDict myDict;

void setup()
{
    String[] words = loadStrings("words");
    for( int idx = 0; idx < words.length; ++idx ) {
        myDict.set( words[idx], 1 );
    }
}

boolean isWord( String word )
{
    return myDict.hasKey( word );
}
```

These are guaranteed to be fast!
# Counting things

<table>
<thead>
<tr>
<th>Absalom, Absalom!</th>
<th>A Farewell To Arms</th>
<th>Alice in Wonderland</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blood Meridian</td>
<td>Frankenstein</td>
<td>Great Expectations</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Huckleberry Finn</td>
<td>Pride and Prejudice</td>
<td>Ulysses</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Finding patterns

It’s easy to search a string for a given phone number:

```java
if( myString.contains( "(519) 888-4567" ) ) { ... }
```

But what if we wanted to find all the phone numbers in a string?
Finding patterns

Regular Expressions are a general tool for finding patterns in strings.
Finding patterns

Regular Expressions are a programming language for finding patterns in strings.
Finding patterns

Regular Expressions are a cryptic programming language for finding patterns in strings.

xkcd.com/208/
Look for an instance of the regular expression pattern inside of the string text. If the answer is not null, the pattern was found.
## Regular Expressions - Quick Reference Guide

### Anchors
- `^` start of line
- `$` end of line
- `\b` word boundary
- `\B` not at word boundary
- `\A` start of subject
- `\G` first match in subject
- `\z` end of subject
- `\Z` end of subject
- `\Q` or before newline at end

### Non-printing characters
- `\a` alarm (BEL, hex 07)
- `\e` escape (hex 1B)
- `\f` formfeed (hex 0C)
- `\n` newline (hex 0A)
- `\r` carriage return (hex OD)
- `\t` tab (hex 09)
- `\v` vertical tab (hex 0B)
- `\xhh` hex code hh
- `\hh` octal code ddd
- `{\hh\.}{\hh\.}` generic character types
- `\d` decimal digit
- `\D` not a decimal digit
- `\s` whitespace character
- `\S` not a whitespace character
- `\w` "word" character
- `\W` "non-word" character

### POSIX character classes
- `alnum` letters and digits
- `alpha` letters
- `ascii` character codes 0-127
- `blank` space or tab only
- `cntrl` control characters
- `digit` decimal digits
- `graph` printing chars - space
- `lower` lower case letters
- `print` printing chars + space
- `punct` printing chars -alnum
- `space` white space
- `upper` upper case letters
- `word` "word" characters
- `xdigit` hexadecimal digits

### Literal Characters
- Letters and digits match exactly `a x B 7 0`
- Some special characters match exactly `@ - = %`
- Escape other specials with backslash `\ \ \ \ ![`

### Character Groups
- Almost any character (usually not newline). `.`
- Lists and ranges of characters `[ ]`
- Any character except those listed `[^ ]`

### Counts (add ? for non-greedy)
- 0 or more ("perhaps some") `*`
- 0 or 1 ("perhaps a") `?`
- 1 or more ("some") `+`
- Between "n" and "m" of `{n,m}
- Exactly "n", "n" or more `{n}, {n,}

### Alternation
- Either/or `|`

### Lookahead and Lookbehind
- Followed by `(?=.* )`
- NOT followed by `(?<! )`
- Following `(?<=.*)`
- NOT following `(?<! )`

### Grouping
- For capture and counts `( )`
- Non-capturing `(?:)`
- Named captures `(?<name> )`

### Back references
- Numbered `
- Relative `g{-n}`
- Named `\k<name>`

### Character group contents
- Individual chars `x`
- Character range `x-y`
- Posix char class `[^class:]`
- Negated class `[^<:class:]

### Examples
- `[a-zA-Z0-9_]
- `[[[:alnum:]]

### Comments
- `(?#comment`

### Conditional subpatterns
- `(?(condition)yes-pattern)`
- `(?(condition)yes!no-pattern)`

### Recursive patterns
- `(?n)` Numbered
- `(?0) (?R)` Entire regex
- `(?&name)` Named

### Replacements
- `$n` reference capture

### Case foldings
- `\u` upper case next char
- `\U` upper case following
- `\l` lower case next char
- `\L` lower case following
- `\E` end case folding

### Conditional insertions
- `(?n:insertion)`
- `(?n:insertion:otherwise)`

[http://www.e-texteditor.com](http://www.e-texteditor.com)
Substring “ufa” anywhere in a word:

ufa

Word ending in “mt”:

mt$

Word with three or more “y”s, on a line by itself:

y.*y.*y

An integer:

^(-?[1-9]+\d*)$|^0$

An email address:

\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}\b

A URL:

^(https?:\/\/)?([da-z\.-]+\.[a-z\.-]{2,6})([\w\.-]*)*$