Module 10

Data Processing and Text

CS 106 Winter 2018
3,855,916,748
Internet Users in the world

1,340,377,646
Total number of Websites

204,637,856,125
Emails sent today

4,829,632,657
Google searches today

4,544,266
Blog posts written today

585,848,322
Tweets sent today

5,369,706,868
Videos viewed today on YouTube

61,276,587
Photos uploaded today on Instagram

99,302,167
Tumblr posts today
TOTAL GLOBAL EMAIL & SPAM VOLUME FOR JANUARY 2018

Average Daily Legitimate Email Volume

73.13 BILLION

Email Volume Change from Previous Month

3.4%

Average Daily Spam Volume

421.81 BILLION

Spam Volume Change from Previous Month

+4%

DAILY EMAIL VOLUME

<table>
<thead>
<tr>
<th>EMAIL TYPE</th>
<th>AVERAGE DAILY VOLUME (BILLIONS)</th>
<th>PERCENTAGE OF GLOBAL TRAFFIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legitimate</td>
<td>80.02</td>
<td>14.70%</td>
</tr>
<tr>
<td>Spam</td>
<td>464.12</td>
<td>85.29%</td>
</tr>
</tbody>
</table>
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>YouTube Company Statistics</th>
<th>Data</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 Instant Video**: 1.61%
- **Facebook**: 1.31%
- **Hulu**: 1.29%
- **Other**: 23.77%

* September 2013. Fixed access only.

Source: Sandvine
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
Make More Sales with Criteo Today.
10,000 brands · 130 countries
Services: Transparent CPC Pricing, Unparalled 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
#1 Marketing Platform® For Advertising Online. Free Demo!

Online Advertising - Reach More Customers Online
Demo OutRank's Powerful Platform.

Online advertising - Wikipedia, the free encyclopedia
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

Delete all spam messages now (messages that have been in Spam more than 30 days will be automatically deleted)

<table>
<thead>
<tr>
<th>Sender</th>
<th>Subject</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>NBA.com</td>
<td>Azriel, NBA Saturday Primetime Continues Tonight on ABC - ABC is the place to</td>
<td>8:04 pm</td>
</tr>
<tr>
<td>MilleniumCard</td>
<td>You may be approved for a $1,000 Credit Card</td>
<td>7:54 pm</td>
</tr>
<tr>
<td>Cannabis.Oil</td>
<td>Welcome to Cannabis-Oil - You Have Been Selected! - Try CBD Oil for Free!</td>
<td>7:27 pm</td>
</tr>
<tr>
<td>Monitor Sex Offenders</td>
<td>Pedophile Alert in your area zyUm - New Sex Offender Update 02/24/2018</td>
<td>6:09 pm</td>
</tr>
<tr>
<td>Amazon.ca</td>
<td>Save 63% today - We have recommendations for you Cliquez ici pour voir vos recor</td>
<td>2:20 pm</td>
</tr>
<tr>
<td>CannablsOil</td>
<td>Test Pure CBD Oil 100% FREE kf - Cannabis Oil Without a Prescription in All 50 St </td>
<td>1:26 pm</td>
</tr>
<tr>
<td><a href="http://www.Madstore.SK">www.Madstore.SK</a></td>
<td>MadStore.SK Best Underground Market - LOGIN BACK NEW DOMAIN REGISTEI</td>
<td>12:36 pm</td>
</tr>
<tr>
<td>Online Concealed Gun</td>
<td>Qualify to carry a Gun_Legally. Start for [F]ree Today..</td>
<td>11:04 am</td>
</tr>
<tr>
<td>Spencer's Online</td>
<td>What Type Are You?  - + Get $20 OFF! Click here to view online Spencer's Save</td>
<td>10:07 am</td>
</tr>
<tr>
<td>Scotty</td>
<td>Please complete your file OQC00Y6Y - Hello Tina, I don't know if you heard the new</td>
<td>9:56 am</td>
</tr>
<tr>
<td>MaxLoan</td>
<td>Cash Advance for Tina Goldsmith - Dear Tina, Get approved for $15000 Immediately</td>
<td>8:56 am</td>
</tr>
<tr>
<td>Thoughtful Mind</td>
<td>Inspiring Quote for February 24th, 2018 - Not receiving in your Inbox? Please add</td>
<td>8:41 am</td>
</tr>
<tr>
<td>Yui</td>
<td>Tina,Desperate Asian Girls Looking for Dates! - Tina,Desperate Asian Girls Lookir</td>
<td>8:38 am</td>
</tr>
<tr>
<td>Liberty Mutual Offer</td>
<td>Liberty Mutual: Ready to start saving.. - This email message contains information</td>
<td>8:10 am</td>
</tr>
<tr>
<td>Cron Daemon</td>
<td>Cron <a href="mailto:root@vps">root@vps</a> /usr/local/cpanel/scripts/upcp --cron - [2018-02-24 03:40:02 -0</td>
<td>7:43 am</td>
</tr>
<tr>
<td>Red Lobster_Free Samples</td>
<td>Get your $100 Red Lobster gift card today - If you wish not to receive these mails</td>
<td>7:18 am</td>
</tr>
<tr>
<td>MaxLoan</td>
<td>Tina,your cash is arrived! - Dear Tina, Get approved for $15000 Immediately Apply</td>
<td>6:25 am</td>
</tr>
</tbody>
</table>
Hang w/
Welcome To Hang w/ - Private Con Man Access...

Humble Bundle
Your Humble Bundle order is ready

Con Man Productions
Con Man Commentary Tracks with Alan Tudor...

Con Man Productions
Stretch Goal Reward: Free Spectrum Issue #0!

Cye Waldman
[The Tiling List] Voderberg rant

XVideos
Your account on Xnxx.com

Official Pokémon GO Team
Videos: Ditto, New Pokémon and Fun Facts for T...

Ubisoft Account Support
Welcome to Ubisoft

FirstMediaX Info
Informasi mengenai Link Account 9995805904...

CanadaHelps
Your charitable tax receipt for 2016 is ready!

Kitchener-Waterloo Symphony
Renew your 2017/18 subscription by March 17 for...

billing@canadianwebhosting.com
Customer Invoice

Official Pokémon GO Team
Videos: Ditto, New Pokémon and Fun Facts for T...
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 Customer Care, at your service.

Community @Amex
Your questions. Your interests. Your community.
1700 Years of Global Temperature Change from Proxy Data

Temperature Change (°F)

- Year
  - 300
  - 500
  - 700
  - 900
  - 1100
  - 1300
  - 1500
  - 1700
  - 1900

- uncertainty
- proxy-based records
- thermometer-based records

- 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.
How to be a Dadaist troll

Inside the Internet Research Agency’s lie machine

Serving up fake news for Mr Putin’s chef
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.
Raw text
Sequence
Dictionary
Table
Tree
Graph
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
McCarthy

Faulkner

medium.com/@neuroecology/punctuation-in-novels-8f316d542ec4
<table>
<thead>
<tr>
<th></th>
<th>A Farewell To Arms</th>
<th>Alice in Wonderland</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absalom, Absalom!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blood Meridian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Huckleberry Finn</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frankenstein</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pride and Prejudice</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Great Expectations</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The values above zero indicate 'positive' emotions, and the values below zero indicate 'negative' emotions.

Red is negative, green is positive, yellow is neutral.
Hi Craig,

Received: from CONNMBX02.connect.uwaterloo.ca ([129.97.149.109]) by connhub1.connect.uwaterloo.ca ([129.97.149.101]) with mapi id 14.03.0319.002; Tue, 17 Jan 2017 15:57:38 -0500
From: Rishabh Moudgil <rishabh.moudgil@uwaterloo.ca>
To: Craig Kaplan <csk@uwaterloo.ca>
CC: Kevin Harrigan <kevinh@uwaterloo.ca>, Kristina Bayda <kabayda@uwaterloo.ca>, Travis Bartlett <travis.bartlett@uwaterloo.ca>
Subject: A01 Marking Scheme
Thread-Topic: A01 Marking Scheme
Thread-Index: AdJw/+DUxNKRRlCRRKOZfc2CQLKSng==
Date: Tue, 17 Jan 2017 20:57:36 +0000
Message-ID: <748888CA42FDF349AF07A8978DDED060281C9EC0@connmbx02>
Accept-Language: en-CA, en-US
Content-Language: en-CA
X-MS-Exchange-Organization-AuthAs: Internal
X-MS-Exchange-Organization-AuthMechanism: 04
X-MS-Exchange-Organization-AuthSource: connhub1.connect.uwaterloo.ca
X-MS-Has-Attach:
X-MS-Exchange-Organization-SCL: -1
X-MS-TNEF-Correlator:
Content-Type: multipart/alternative;
boundary="_000_748888CA42FDF349AF07A8978DDED060281C9EC0connmbx02_"
MIME-Version: 1.0

--_000_748888CA42FDF349AF07A8978DDED060281C9EC0connmbx02_
Content-Type: text/plain; charset="Windows-1252"
Content-Transfer-Encoding: quoted-printable

Hi Craig,
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>
</tr>
</thead>
<tbody>
<tr>
<td>1896</td>
<td>Athens, Greece</td>
</tr>
<tr>
<td>1900</td>
<td>Paris, France</td>
</tr>
<tr>
<td>1904</td>
<td>St. Louis, United States</td>
</tr>
<tr>
<td>1908</td>
<td>London, United Kingdom</td>
</tr>
<tr>
<td>1912</td>
<td>Stockholm, Sweden</td>
</tr>
<tr>
<td>1920</td>
<td>Antwerp, Belgium</td>
</tr>
<tr>
<td>1924</td>
<td>Paris, France</td>
</tr>
<tr>
<td>1928</td>
<td>Amsterdam, Netherlands</td>
</tr>
<tr>
<td>1932</td>
<td>Los Angeles, United States</td>
</tr>
<tr>
<td>1936</td>
<td>Berlin, Germany</td>
</tr>
<tr>
<td>1948</td>
<td>London, United Kingdom</td>
</tr>
<tr>
<td>1952</td>
<td>Helsinki, Finland</td>
</tr>
<tr>
<td>1956</td>
<td>Melbourne, Australia</td>
</tr>
<tr>
<td>1960</td>
<td>Rome, Italy</td>
</tr>
<tr>
<td>1964</td>
<td>Tokyo, Japan</td>
</tr>
<tr>
<td>1968</td>
<td>Mexico City, Mexico</td>
</tr>
<tr>
<td>1972</td>
<td>Munich, West Germany</td>
</tr>
<tr>
<td>1976</td>
<td>Montréal, Canada</td>
</tr>
<tr>
<td>1980</td>
<td>Moscow, Soviet Union</td>
</tr>
<tr>
<td>1984</td>
<td>Los Angeles, United States</td>
</tr>
<tr>
<td>1988</td>
<td>Seoul, South Korea</td>
</tr>
<tr>
<td>1992</td>
<td>Barcelona, Spain</td>
</tr>
<tr>
<td>1996</td>
<td>Atlanta, United States</td>
</tr>
<tr>
<td>2000</td>
<td>Sydney, Australia</td>
</tr>
<tr>
<td>2004</td>
<td>Athens, Greece</td>
</tr>
<tr>
<td>2008</td>
<td>Beijing, China</td>
</tr>
<tr>
<td>2012</td>
<td>London, United Kingdom</td>
</tr>
<tr>
<td>2016</td>
<td>Rio de Janeiro, Brazil</td>
</tr>
<tr>
<td>2020</td>
<td>Tokyo, Japan</td>
</tr>
<tr>
<td>SONG</td>
<td>ARTIST</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Ways To Go - Margot Mix</td>
<td>Weval, Margot</td>
</tr>
<tr>
<td>Death Is A Girl</td>
<td>Mini Mansions</td>
</tr>
<tr>
<td>Jumbo</td>
<td>Underworld</td>
</tr>
<tr>
<td>Bug Powder Dust</td>
<td>The Mysterons</td>
</tr>
<tr>
<td>...To Have No Answer</td>
<td>Flock of Dimes</td>
</tr>
<tr>
<td>I'll Cut You Down</td>
<td>Uncle Acid &amp; The...</td>
</tr>
<tr>
<td>L'enfer ce n'est pas les autres c'est moi</td>
<td>The Eye Of Time</td>
</tr>
<tr>
<td>Terrain</td>
<td>pg_lost</td>
</tr>
</tbody>
</table>
save()
saveFrame()

Files
beginRaw()
beginRecord()
createOutput()
createWriter()
endRaw()
endRecord()

Functions
abs()
cell()
constrain()
dist()
exp()
floor()
lerp()
log()
mag()
map()
max()
min()
norm()
pow()
round()
sgn()
sqrt()

Trigonometry
acos()
asin()
atan()
atan2()
cos()
degrees()
radians()
sin()
tan()

Transform
applyMatrix()
popMatrix()
printMatrix()
pushMatrix()
resetMatrix()
rotate()
rotateX()
rotateY()
rotateZ()
scale()
shearX()
shearY()
translate()

Random
noise()
oiseDetail()
noiseSeed()
random()
randomGaussian()
randomSeed()
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

```
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 );

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 );

Check if two strings have the same characters
String operations

String \( wd = "..." \);

int \( \text{len} = \text{wd} . \text{length}() \);

char \( c = \text{wd} . \text{charAt}(2) \);

String \( \text{str3} = \text{str1} + \text{str2} \);

if( \( \text{str1} . \text{equals}( \text{str2} ) \) ) { ... }

String[] \( \text{words} = \text{splitTokens}( \text{str1} ) \);

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

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>Addley</td>
<td>Cambridge</td>
</tr>
<tr>
<td>Addy</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

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

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

if( str.contains( "abc" ) ) { … }

Check if the string \texttt{str} has the substring \texttt{“abc”} anywhere inside of it.

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!
The function `join()` is like the reverse of `splitTokens()`: it turns an array of strings into one long string, using a given delimiter string.

```java
String[] things = {
    "Kumquat", "Durian", "Rambutan", "Lychee" 
};

println( join( things, " " ) );

println( join( things, " and " ) );
```
The function `join()` is like the reverse of `splitTokens()`: it turns an array of strings into one long string, using a given delimiter string.

```java
String[] things = {
    "Kumquat", "Durian", "Rambutan", "Lychee" }

println( join( things, " ") );
 ⇒ Kumquat Durian Rambutan Lychee
println( join( things, " and " ) );
```
The function `join()` is like the reverse of `splitTokens()`: it turns an array of strings into one long string, using a given delimiter string.

```java
String[] things = {
    "Kumquat", "Durian", "Rambutan", "Lychee"};

println( join( things, " ") );
⇒ Kumquat Durian Rambutan Lychee

println( join( things, " and ") );
⇒ Kumquat and Durian and Rambutan and Lychee
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
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
Create a new, empty dictionary

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

Look up the value associated with a given key

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

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 myDictHasKey( word );
}
Writing a spellchecker

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 );
}
## 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>?</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Blood Meridian</th>
<th>Frankenstein</th>
<th>Great Expectations</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>!</td>
<td>!</td>
</tr>
<tr>
<td>?</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Huckleberry Finn</th>
<th>Pride and Prejudice</th>
<th>Ulysses</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>!</td>
<td>!</td>
</tr>
<tr>
<td>?</td>
<td></td>
<td></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/
String[] match(String text, String pattern) {
    ... }

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
- word boundary
- not at word boundary
- start of subject
- first match in subject
- end of subject
- end of subject or before newline at end

## Non-printing characters
- \a alarm (BEL, hex 07)
- \b "control-b"
- \e escape (hex 1B)
- \f formfeed (hex 0C)
- \n newline (hex 0A)
- \r carriage return (hex OD)
- \t tab (hex 09)
- \v vertical tabulation (hex 0B)
- \xhh octal code ddd
- \xhh hex code hh
- \{hhh.} hexadecimal digits

## 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 lowercase letters
- print printing chars +space
- punct printing chars -alnum
- space white space
- upper uppercase letters
- word "word" characters
- xdigit hexadecimal digits

## Literal Characters
- Letters and digits match exactly: a 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: \1 \2 \3
- Relative: \g{-n}
- Named: \k<name>

## Character group contents
- x individual chars
- x-y character range
- [class:] posix char class
- [^class:] negated 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)
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\})([\da-zA-Z\.-]+)*$
A regular expression is like a little “machine”:

\(^{-?\[1-9]\+\d\ast}\) | \(^0\)