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
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 or other, cherish very nearly the same feelings towards the ocean with me.
Hi Craig,

I hope you're having a great day!

Written below is a rough draft of A01:

---

Content-Type: text/plain; charset="Windows-1252"
Content-Transfer-Encoding: quoted-printable
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 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
# 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>
Tree
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 );

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 );

Glue two strings together
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>
</tbody>
</table>

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 repeated letters

A
a
aa
aal
aalii
aam
Aani
aardvark
aardwolf
Aaron
Aaronic
Aaronical
Aaronite
Aaronitic
Aaru
Ab
aba
Ababdeh
Ababua
abac
abaca
abacate
abacay
abacate
Finding things in strings

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

Check if the string str has the substring “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;
}
```

With the dictionary at our disposal, it’s easy to check if a given string is a word.
boolean isWord( String word ) {
    for ( int idx = 0; idx < dict.length; ++idx ) {
        if ( dict[idx].equals( word ) ) {
            return true;
        }
    }
    return false;
}
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
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
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.containsKey("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.containsKey( word );
}
```
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 );
}
```

Dict is guaranteed to be fast!
Counting things
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
- \b word boundary
- \B not at word boundary
- \a start of subject
- \G first match in subject
- \z end of subject
- \G end of subject or before newline at end

### Non-printing characters
- \a alarm (BEL, hex 07)
- \cx "control-x"
- \e escape (hex 1B)
- \f formfeed (hex 0C)
- \n newline (hex 0A)
- \r carriage return (hex OD)
- \t tab (hex 09)
- \dddd octal code dddd
- \xhh hex code hh
- \x{hhh} hex code hhh

### Generic character types
- \d decimal digit
- \D not a decimal digit
- \s whitespace character
- \S not a whitespace char
- \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: 
- Exactly "n", "n" or more:

### Alternation
- Either/or

### Lookahead and Lookbehind
- Followed by
- NOT followed by
- Following
- NOT following

### Grouping
- For capture and counts:
- Non-capturing:
- Named captures:

### Back references
- Numbered
- Relative
- Named

### Character group contents
- x individual chars
- x-y character range
- [class:] character 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]{2,6}(\/[\w \.-]*)*\/?$

A URL:
    ^(https?:\/\/)?([^da-z.-]+\.[^a-z.]{2,6})([\/w \.-]*)*\/?$
A regular expression is like a little “machine”:

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