There's a simple reason your new smart TV was so affordable: It's collecting and selling your data

“If you want a 65-inch 4K smart TV with HDR capability, one can be purchased for below $500. But that low price comes with a caveat most people probably don't realize: Some manufacturers collect data about users and sell that data to third parties.

“Smart TVs can be sold at or near cost to consumers because Vizio is able to monetize those TVs through data collection, advertising, and selling direct-to-consumer entertainment (movies, etc.).”

Administrivia

- Assignments:
  - Arrays and Elli [checkoff] due tomorrow (2/14)
  - Reading Check 5 due @ 3:30 pm tomorrow (2/14)
  - Color Filters [checkoff] due on Tuesday (2/19)
  - Controlling Elli [submit] due on Tuesday (2/19)
  - Living Computers Museum Report due in 2 weeks (2/26)

- Guest lecture on Friday: Artificial Intelligence
Field trip out to the Living Computers: Museum + Labs in SoDo

- Admission is paid for you!
- Transportation: Link + walk, bus, drive
- Go when you can: open Wed-Sun each week

Report: PDF including photos and responses due 2/26

- Part 1: Favorite Exhibit
- Part 2: Computer History
- Part 3: Modern Tech Exhibit Reflection
Outline

- Images
- Compression
- Strings
Images

- An image is just a 2-dimensional set of pixels
  - The image has a width and a height
  - Each pixel has an associated (RGB) color
Images

- An image is just a 2-dimensional set of pixels
  - The image has a width and a height
  - Each pixel has an associated (RGB) color

- In Processing, an image is represented as an array of color data
  - Can explicitly use `color[]` myImage
  - Processing also provides special datatype `PImage`
Using Images in Processing

1) Load an image from a file into a Processing variable
   ▪ Use the `loadImage("photo.jpg")` function
     • The image name is a String representing the *path* to the file, similar to your website
   ▪ Store the return value from `loadImage()` into a `PImage` variable
     • *e.g.* `PImage myImg = loadImage("img/justin.jpg");`

2) Draw the image on your canvas using the `image()` function
   ▪ `image(<PImage var>, <x_pos>, <y_pos>)`
   ▪ *e.g.* `image(myImg, 0, 0);`
The Canvas as an Image

- The drawing canvas is also treated as an image!
  - Retrieve the current canvas image data (i.e. array of color data) using the `loadPixels()` function
    - `loadPixels()` has no parameters or return value
    - The canvas image data will be automatically stored into the system variable `pixels[]`
  - You can manually manipulate the data in `pixels[]`
    - *e.g.* `pixels[0] = color(0);` // set to black
  - Update the drawing canvas with the current/new data in `pixels[]` using the `updatePixels()` function
    - `updatePixels()` also has no parameters or return value
Linearizing an Image

- Despite being 2-D in nature (i.e. x- and y-coordinates), we deal with image data in a 1-D array (i.e. `pixels[]`). The length n uses indices 0 to n-1.
  - As we increment our array index, we move left-to-right horizontally and then top-to-bottom vertically.

```
<table>
<thead>
<tr>
<th>start</th>
<th>width (16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>mouseY</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 1</td>
</tr>
<tr>
<td>row 2</td>
</tr>
<tr>
<td>row 3</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>mouseX</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 1</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>row 2</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>row 3</td>
</tr>
<tr>
<td>48</td>
</tr>
</tbody>
</table>
```

```
index = row * width + col
```

- Example: 3 * 16 + 7 = 48 + 7 = 55

```
0 1 2 3 ... 15 = width - 1
```

```
row 3, col 7 <-> index 55
```
Color as Data in Processing

- **Recall:** all data on a computer is stored using *binary encoding*
  - Including colors, though we won’t cover exactly how

- Processing has a special `color` datatype
  - We’re used to using the `color(R, G, B)` function to specify colors
  - Represents colors but looks nonsensical if you try to print it
  - Can retrieve the RGB triplet values using the functions `red()`, `green()`, and `blue()`
Color Filters

- Learn the basics of using and manipulating images in Processing
  - You choose a photo to display
  - Display the RGB of the pixel your mouse is hovering over
  - Key presses will filter the colors of your image appropriately
Outline

- Images
- Compression
- Strings
Compression

Compression is the process of encoding information/data using fewer bits than the original representation

- **Lossless**: original bits can be *exactly* recovered from transformed bits
- **Lossy**: original bits *cannot* be exactly recovered from transformed bits (*i.e.* some data is lost)
Lossless Compression

- Eliminates bits that can be recovered again
- Consider this 6 x 6 black-and-white image:

- Uncompressed:
  - WWWWWW  BBBBBBB  BWWWWB  BWWWWB  BBBBBBB  WWWWWW
Lossless Image Format: RLE

- Run Length Encoding
  - Not used commonly, but found in formats (TIFF and Bitmap)
  - For repeated data/color, encode # of repeats
  - Many variations on actual encoding exist

- Black-and-white example:
  - 6W 7B 4W 2B 4W 7B 6W

- Flag example:
  - HU = 45:R,45:W,45:G
Lossless Image Format: GIF, PNG

- **Graphics Interchange Format**
  - Uses a 256-color palette (not RGB) encoded in a Color Table
    - Why GIFs may not seem like “true color”
  - Uses **LZW Encoding** (Lempel-Ziv-Welch)
    - Create encodings based on strings of colors in image
    - Supplanted RLE for lossless compression

- **Portable Network Graphics**
  - Improved, non-patented replacement for GIF
  - Doesn’t support animations
Lossy Image Format: JPEG/JPG

- Joint Photographic Experts Group
  - Tradeoff between amount of compression and image quality
  - Areas of similar color are represented by a single shade
    - Based on quantization of discrete cosine transform (DCT) operation
Outline

- Images
- Compression
- Strings
Strings

- A **string** is a string of characters (0 or more)
  - Strings cannot be modified, but string variables can be reassigned
  - Individual characters can be accessed (not modified), numbered from left-to-right *starting at 0* (letters, numbers, symbols, spaces)

- **String literal**: an unnamed string specified between double-quotes
  - *e.g.* "**hello**", "!@#$%^&*()_+ ?~", "**xoxo <3**"
  - "" is known as the **empty string** (0 characters in it)
Using Strings

- **Declaration**: `String str;`

- **Assignment**: `str = "hello";`
  
  - Get character using `str.charAt(i)`
  - Get length using `str.length()`

- **Concatenation**: join strings using `'+` operator
  
  - e.g. "hi " + "there" gives you "hi there"

- Conversion to string usually occurs implicitly
  
  - Can also explicitly use `str()`
Strings vs. Arrays

- **Strings are *sort of* like arrays of characters:**

<table>
<thead>
<tr>
<th></th>
<th>Array</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declare</td>
<td><code>char[] chArray;</code></td>
<td><code>String str;</code></td>
</tr>
<tr>
<td>Initialize</td>
<td><code>chArray = {'h', 'i', '!'};</code></td>
<td><code>str = &quot;hi!&quot;;</code></td>
</tr>
<tr>
<td>Get element</td>
<td><code>chArray[1]</code></td>
<td><code>str. charAt (1)</code></td>
</tr>
<tr>
<td>Get length</td>
<td><code>chArray.length</code></td>
<td><code>str.length()</code></td>
</tr>
</tbody>
</table>
Example: Recording User Input

- `keyPressed()` lets you read user input 1 character at a time

- Use a `String` variable to “store”
  - Add/append new characters using concatenation
Example: Recording User Input

- keyPressed() lets you read user input 1 character at a time

- Use a String variable to “store”
  - Add/append new characters using concatenation

```java
String input = ""; // start with empty string

void draw() {
}

void keyPressed() {
    input = input + str(key);
    println("input = " + input);
}
```

- Convert char to String
- String literal
- Concatenation
Word Guessing

- Learn to use text input & output
  - Player 1 enters a secret phrase
  - Player 2 tries to guess the secret phrase
  - Game tells you how many letters correct & # of attempts