Module 02

Containers

(Arrays, Records, Strings)

Containers

A container is a data element that we treat both as one whole, and as many parts. Like an array.

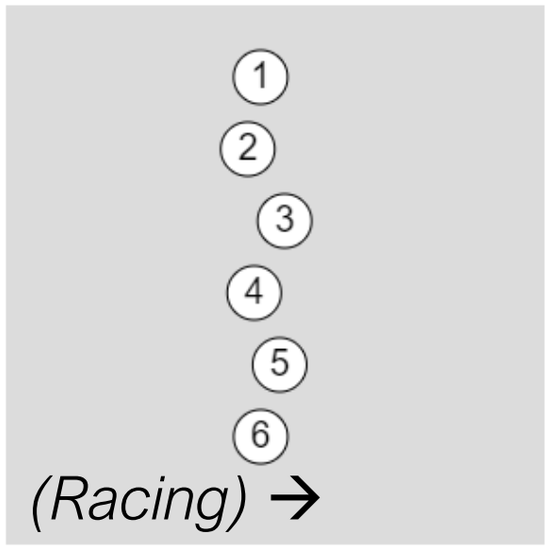
How many boxes does it take to run a business?

How many arrays does it take to code a program?

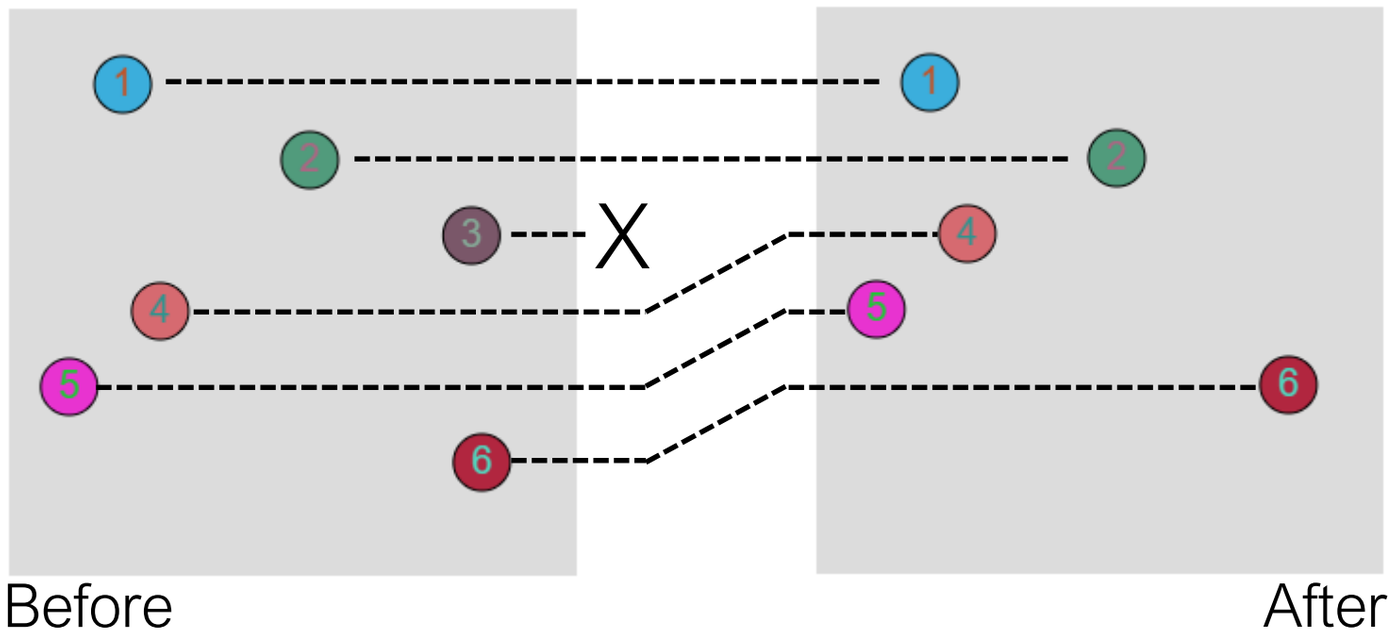
Records

Recall CS 105 circleracers

Then:



Now:



Disqualification!

Records

- A container where elements are accessed by name. Arrays access them by number.

- A way to group variables into folders.

- Common case: array of records. This will be the circleracer trick.

Building blocks: array, record

```
let arr1 = [];

let arr2 = [17, 3 ];

let arr3 = [5 ];
```

```
arr2[1] = arr3[0];
```

```
let n = arr2.length;
```

```
let rec1 = {};

let rec2 = {a: 17, b: 3};

let rec3 = {x: 5};
```

```
rec2.b = rec3.x;
```

```
// no equivalent
```

Building blocks: array, record

No requirement to learn/use this syntax in 106. But you will see it in starter code and in "print" output.

```
let rec1 = {};
let rec2 = {a: 17, b: 3};
let rec3 = {x: 5};

rec2.b = rec3.x;
```

Need to know:

- *empty record*
- *write to b*
- *read from x*

Building blocks: array, record

Overwrites element at position 1 if it was already there; adds it there otherwise.

with name b

```
arr2[1] = arr3[0];
```

```
rec2.b = rec3.x;
```

```
let n = arr2.length;
```

good for looping with zero, defines front/back

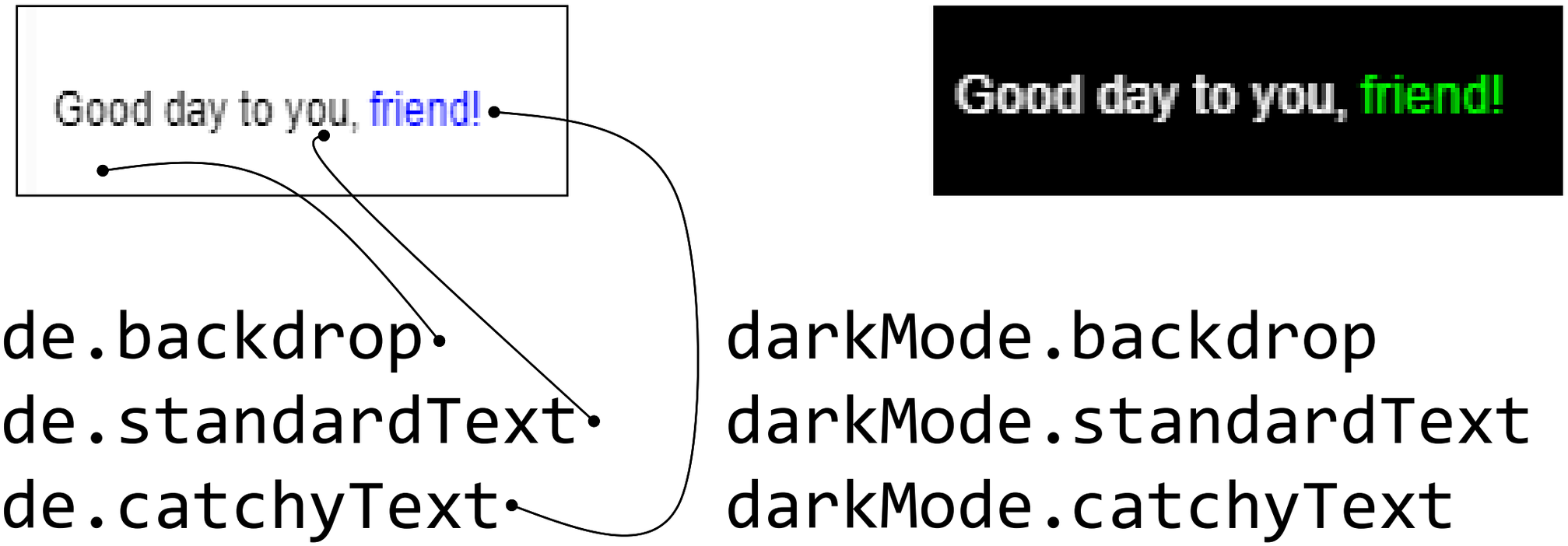
```
// no equivalent
```

- *we don't loop records in 106*
- *record has no order*

Which one prints “Containers?”

Building records

Similar idea to using the color() function. It packages the R, G, B values that you give it into a container.
Let's name some colours.



```
let backdrop;
let standardText;
let catchyText;

function initColors() {

  backdrop = color(255);
  standardText = color(0);
  catchyText = color(0, 0, 255);

}

function setup() {
  createCanvas(200, 100);
  initColors();

}
```

```
let lightMode;
let darkMode;
let curMode;

function initColors() {
  lightMode = {};
  lightMode.backdrop     = color(255);
  lightMode.standardText = color(0);
  lightMode.catchyText   = color(0, 0, 255);
  darkMode = {};
  darkMode.backdrop      = color(0);
  darkMode.standardText  = color(255);
  darkMode.catchyText    = color(0, 255, 0);
}

function setup() {
  createCanvas(200, 100);
  initColors();
  curMode = lightMode;
}
```

```
function draw() {
  background(backdrop);

  fill(standardText);
  text("Good day to you,", 5, 50);

  fill(catchyText);
  text("friend!", 99, 50);
}
```

```
function keyTyped() {
  if (random(0,1) > 0.5) {
    curMode = lightMode;
  } else {
    curMode = darkMode;
  }
}

function draw() {
  background(curMode.backdrop);

  fill(curMode.standardText);
  text("Good day to you,", 5, 50);

  fill(curMode.catchyText);
  text("friend!", 99, 50);
}
```

Recall from CS 105:

p5* circleracers (with array and loop)

```
let x = []; // array

function setup() {
 // initialize 6 racer starting positions
 for (let i = 0; i < 6; i++) {
    x[i] = 0;
 }
}

function draw() {
   ...
   for (let i = 0; i < x.length; i++) {
     // calculate the y position
     let y = 40 * (i + 1);
     racerDraw(x[i], y, 30, i + 1);
     // update the racer's position
     x[i] += random(0, 3);
   }
}
```

New @106:
Add features.
See duplicate code.
Fix with records.

circleracers: First New Feature

- When the user types a number on the keyboard (0–9) the racer in that lane is disqualified

  (and also, so we can see what's happening)
- They go slower
- They reset to the left when they reach the right
- They start with random progress…it's a relative race

circleracers v2: Starting Point

All the “so we can see” features are done.

```
function keyTyped() {
  … disqualifyAt(correctIndex);
}
```

```
function disqualifyAt(i) {
  // we will implement
  print("racer at position " + i + " to be removed");
}

// modifies arr to remove an item that was initially in arr[i];
// shifts later elements forward, and reduces arr.length, by one
function removeElementAt(arr, i) { … }
```

circleracers: problem

- Naïve: just delete from the xs. Remaining racers get lower number after deletion.

- Fix: track the labels in a second array. Remove disqualified racer info from both arrays.

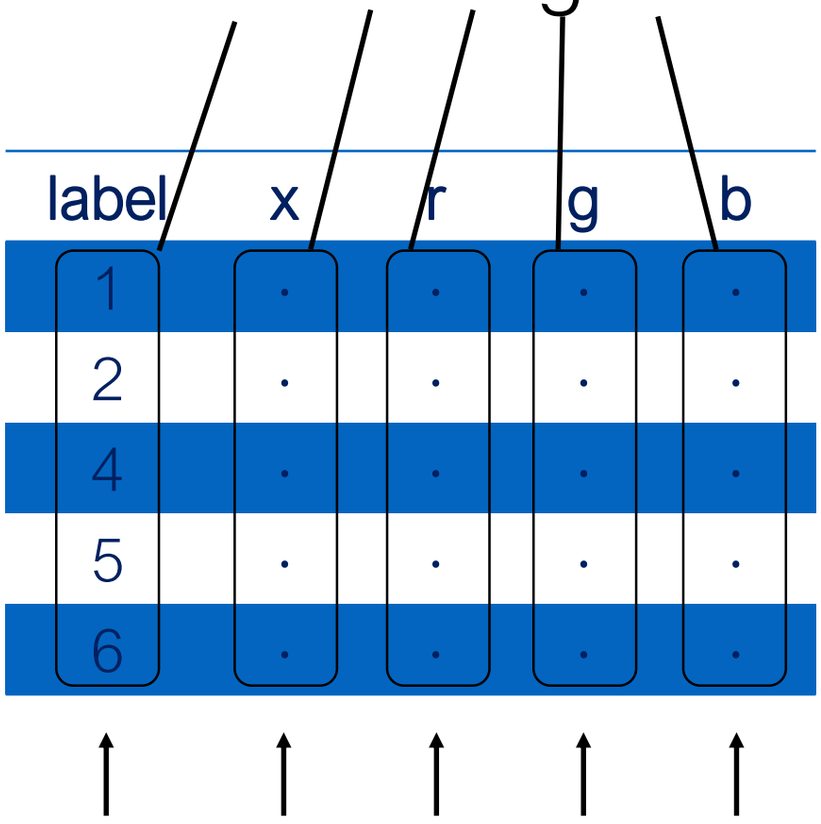
- Next feature: each racer gets a random colour.

```
let x = []; let label = []; let r = []; let g = []; let b = [];
```

- 5 calls to removeElementAt. Each time does “same” shuffling. Your code repeats.

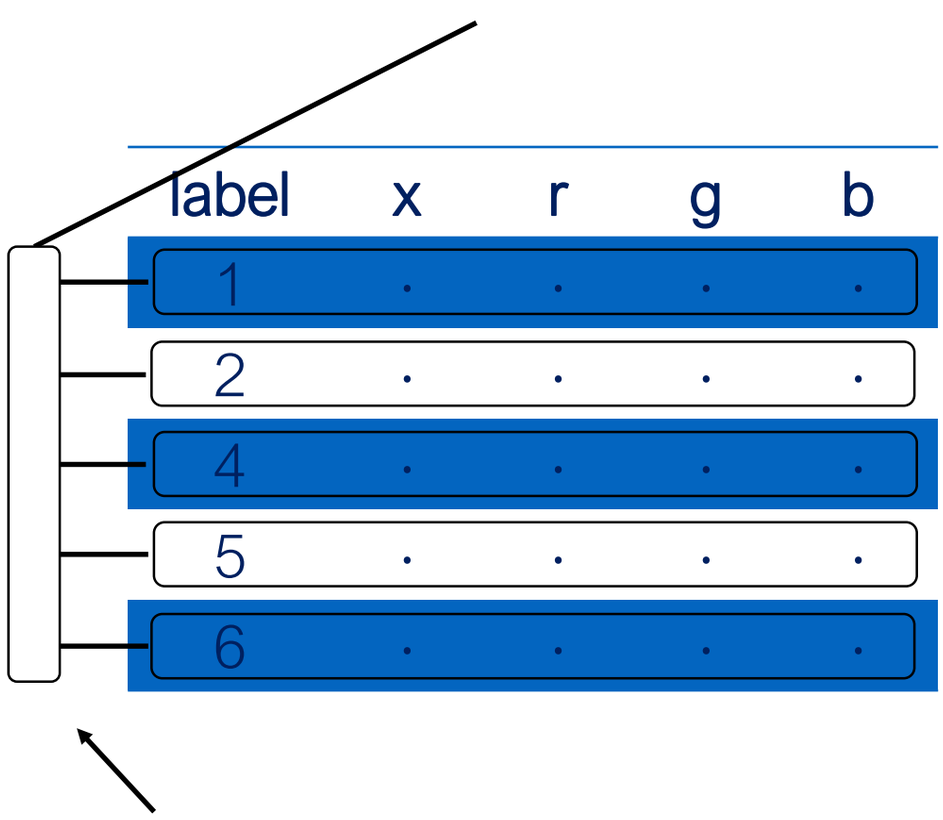
circleracers: solution

Program's variables:



Add, remove, re-order:
Do same on each!

Program's variables:



Add, remove, re-order:
Do once!

Objects by Reference

Objects by Reference

Arrays and records are examples of:

JavaScript P5 **Objects**

JavaScript P5 knows they can get big.
Copying big things can make our program slow.

It wants to help us save on copying.
Here is its “solution.” This pill may cause vertigo.

We’ll work through this with arrays.
It’s the same with records.

What does this print?

What does this print?

Arrays are just values…

```
let arr1 = [1, 2, 3, 4];
let arr2 = arr1;
```

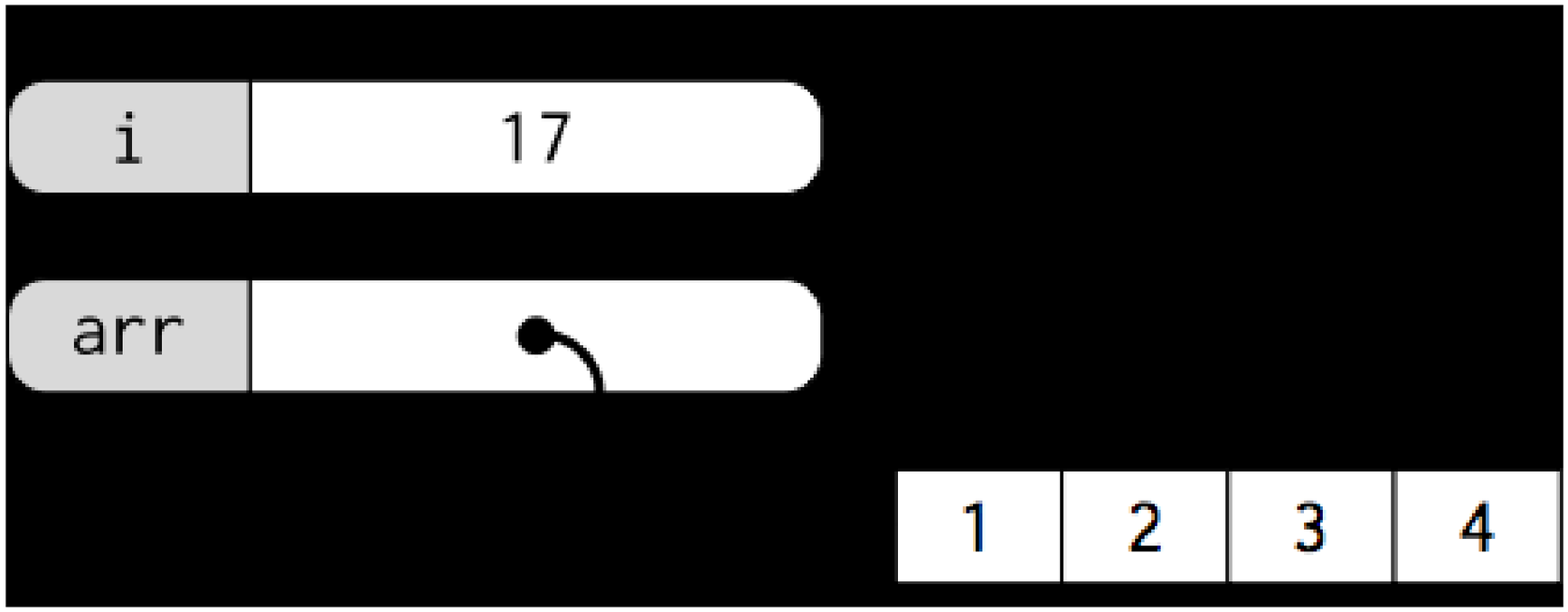
```
// arr is an array, val is a float
function processArray(arr, val) {
  ...
  return arrNew;
}

let arr3 = processArray(arr1, 3.14);
```

…aren’t they?

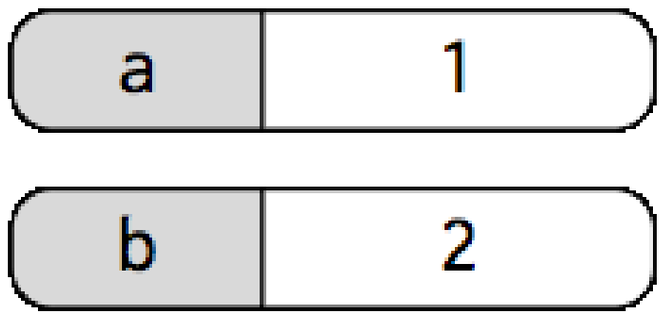
An array value is really an arrow pointing to the place in memory where all the array elements are stored. We say that an array variable is a *reference*.

```
let i = 17;
let arr = [1, 2, 3, 4];
```

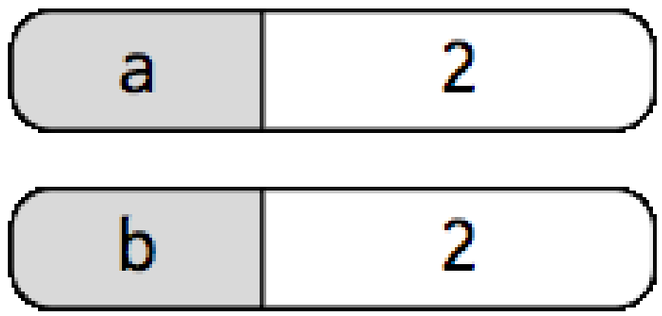


```
let a = 1;
```

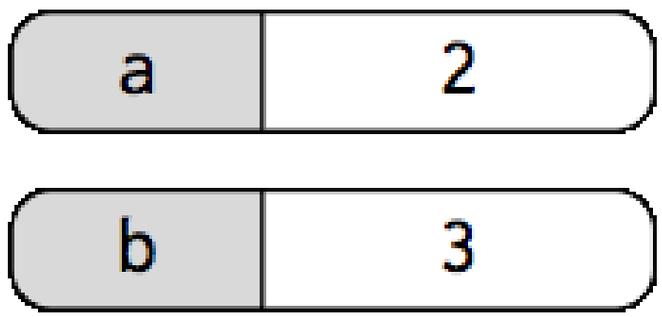
```
let b = 2;
```



```
a = b;
```



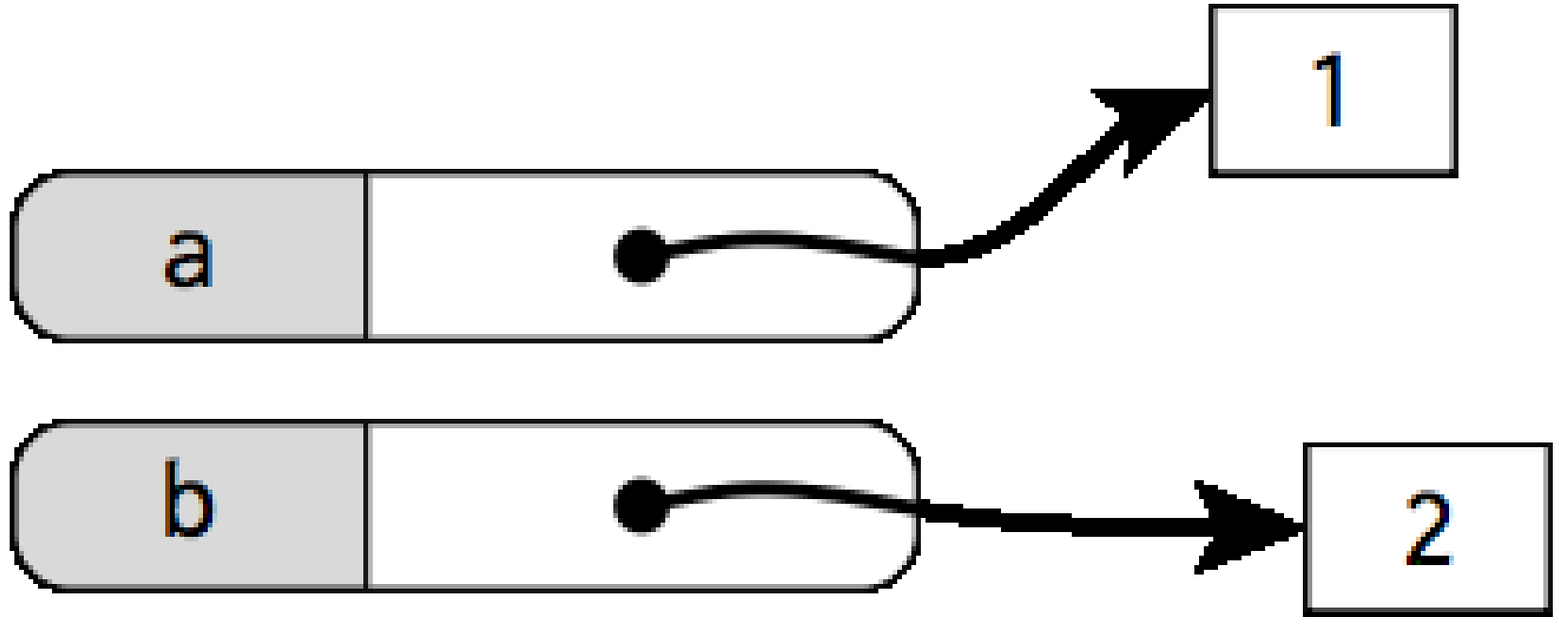
```
b = 3;
```



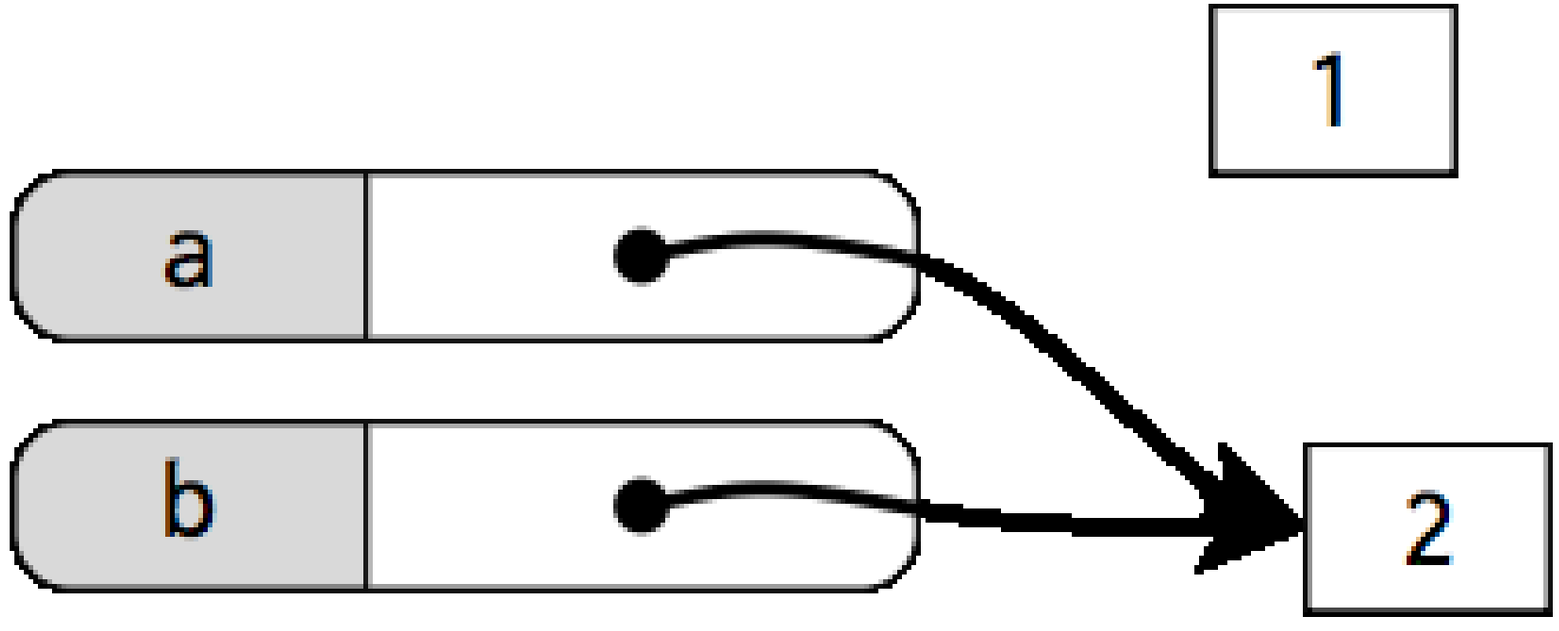
```
let a = [1];
let b = [2];
a = b;
b[0] = 3;
print(a[0] + b[0]);
```

```
let a = [1];
let b = [2];
a = b;
b[0] = 3;
print(a[0] + b[0]);
```

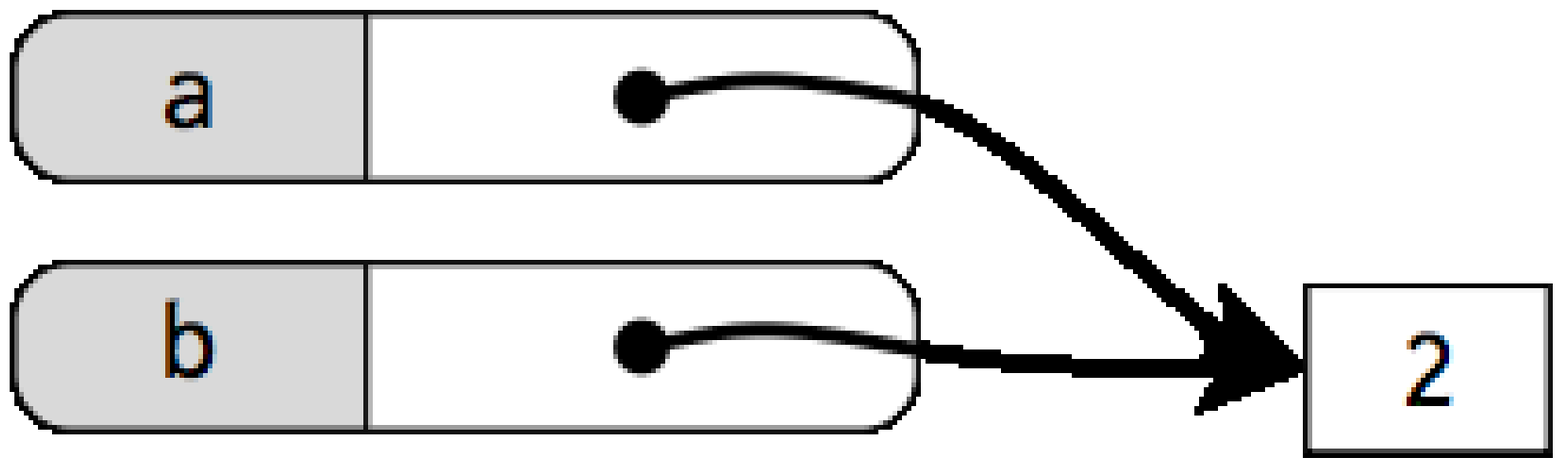
```
let a = [1];
let b = [2];
a = b;
b[0] = 3;
print(a[0] + b[0]);
```



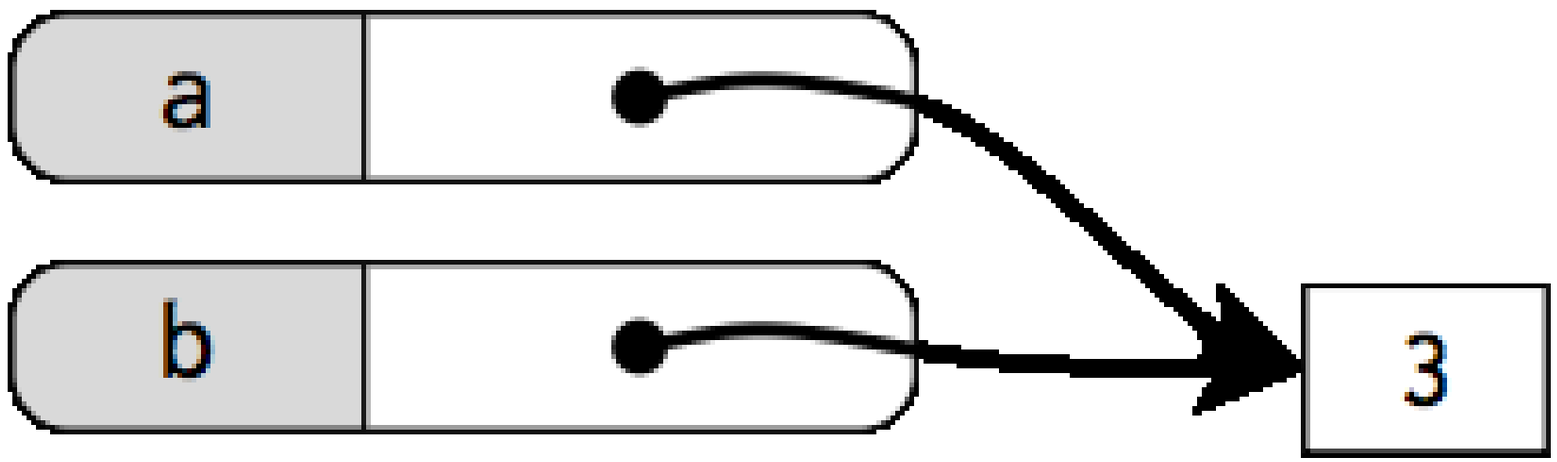
```
let a = [1];
let b = [2];
a = b;
b[0] = 3;
print(a[0] + b[0]);
```



```
let a = [1];
let b = [2];
a = b;
b[0] = 3;
print(a[0] + b[0]);
```



```
let a = [1];
let b = [2];
a = b;
b[0] = 3;
print(a[0] + b[0]);
```



References

The basic types Number and Boolean are “primitive”: their values are “naked” and copied around directly.

Object types (including arrays and records) are passed around by reference (arrows).

What does this print?

Functions on Arrays

Recall Array idioms

An idiom is not a single algorithm or line of code. It's a rough template that can be customized to a specific situation.

```
for (let i = 0; i < arr.length; i++) {
    …
    …
}
```

We can use more than one at once.
We can riff on them.
We should know them, recognize them.

Array functions

Often, a function wraps a case of an idiom.

```
function handleWidgets(arr, …) {
  for (let i = 0; i < arr.length; i++) {
    …
    …
  }
  return something;   ← (maybe)
}
```

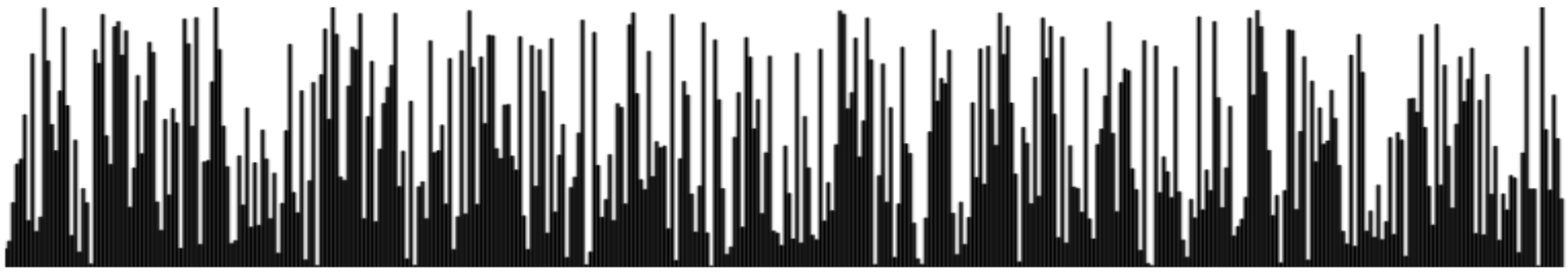
Its parameters and return value (if any) are really given by the idiom too.

Array idioms

1. Item Consumption

"Do" the same action per element. Action like draw.
Well seen in CS 105.

```
// visualize each element as a thin vertical bar
for (let i = 0; i < arr.length; i++) {
  line(i, height, i, height - arr[i]);
}
```

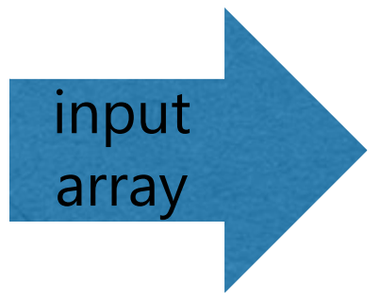


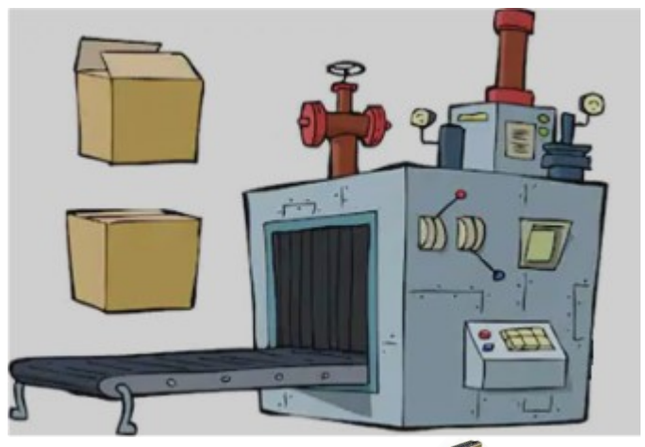
Array functions

1. Item Consumption

"Do" the same action per element. Action like draw.

```
function drawBars(arr) {
  for (let i = 0; i < arr.length; i++) {
    line(i, height, i, height - arr[i]);
  }
  // no return
}
```





Array idioms

2. Distillation

"Reduce" the array down to a single value.
Well seen in CS 105.

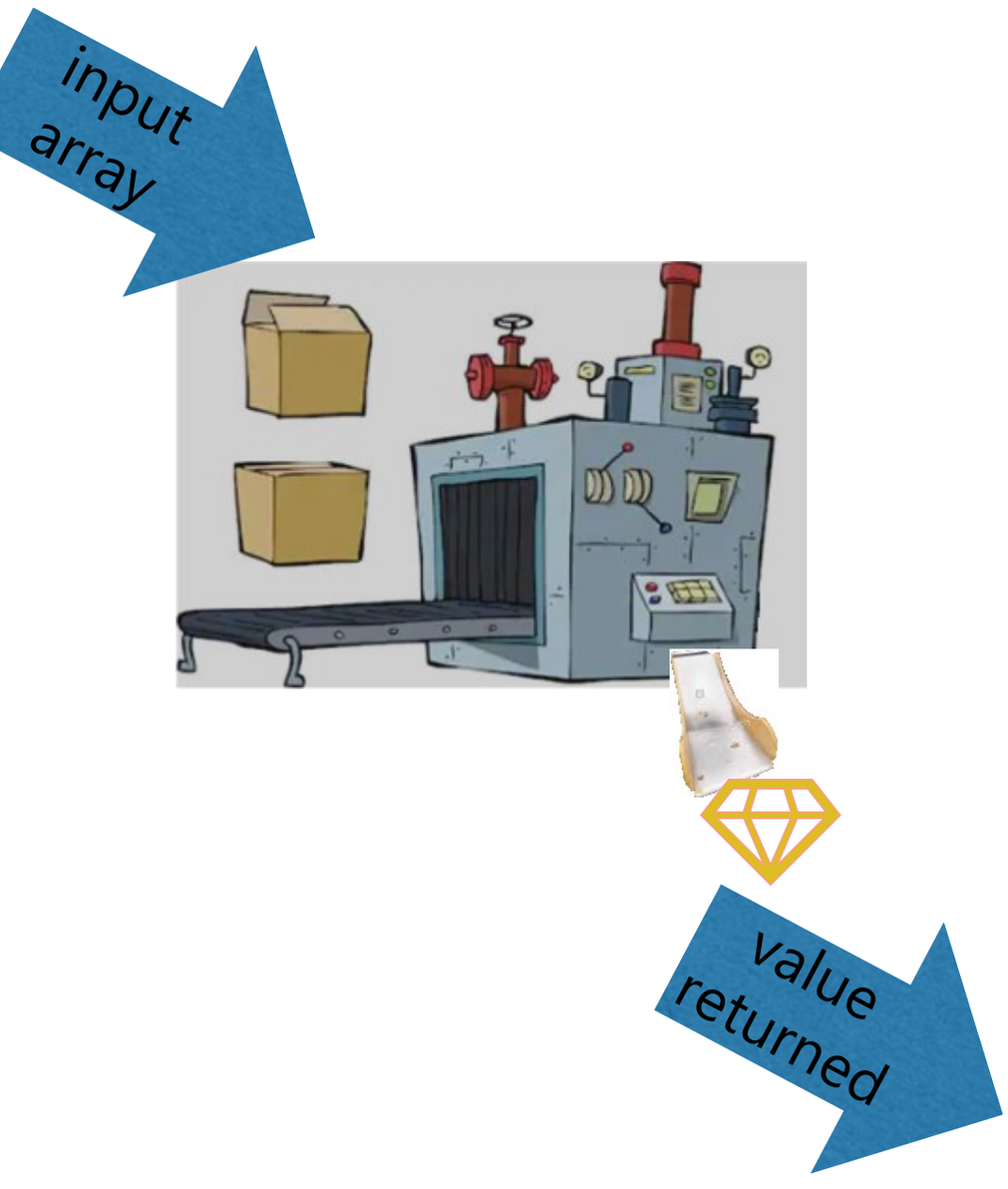
- Largest element
- Smallest element
- Is X in the array?
- Find the index of X
- Sum of elements
- Average of elements
- Number of positive elements

```
let largest = arr[0];
for (let i = 1; i < arr.length; i++) {
  if (arr[i] > largest) {
    largest = arr[i];
  }
}
```

Array functions

2. Distillation

"Reduce" the array down to a single value.




```
function largestElement(arr) {
  let largest = arr[0];

  for (let i = 1; i < arr.length; i++) {
    if (arr[i] > largest) {
      largest = arr[i];
    }
  }

  return largest;
}
```

Array idioms

3. Generation

"Conjure" an array from nothing (or a simple value). Well seen in CS 105.

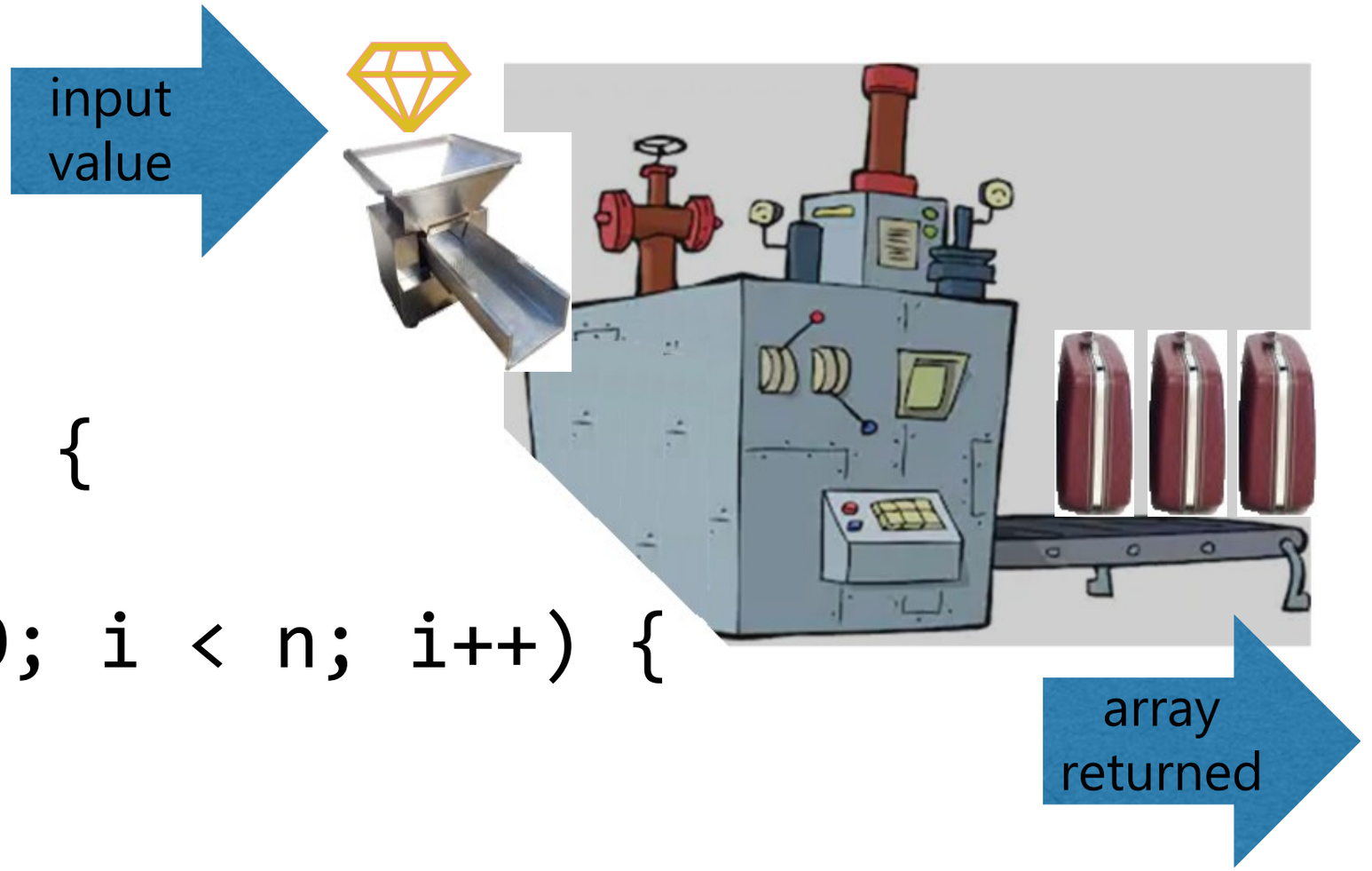
Example: given an integer n, produce the integer array [0, 1, 2, …, n-1].

```
let arr = [];
for (let i = 0; i < n; i++) {
  arr[i] = i;
}
```

Array functions

3. Generation

“Conjure” an array from nothing (or a simple value).




```
function upto(n) {
  let arr = [];
  for (let i = 0; i < n; i++) {
    arr[i] = i;
  }
  return arr;
}
```

Array idioms

4. Item Transformation

"Apply" a smaller transform, across all elements.
May be less familiar.

Example, given a list of contact info, extract just the phone numbers into a new array.

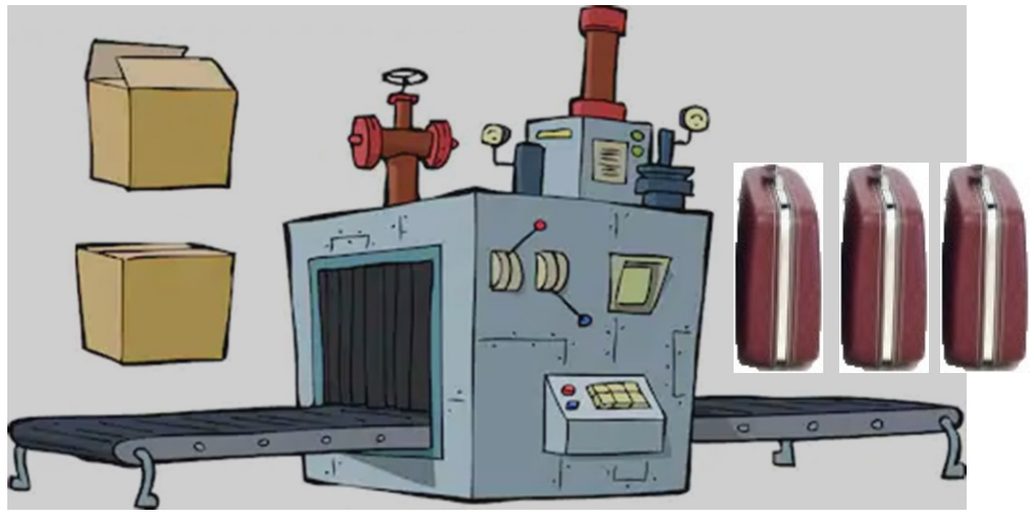
```
let phoneNumbers = [];
for (let i = 0; i < contacts.length; i++) {
  phoneNumbers[i] = contacts[i].phone;
}
```

Array functions

4. Item Transformation

"Apply" a smaller transform, across all elements.

input array



array returned

maybe of something different!

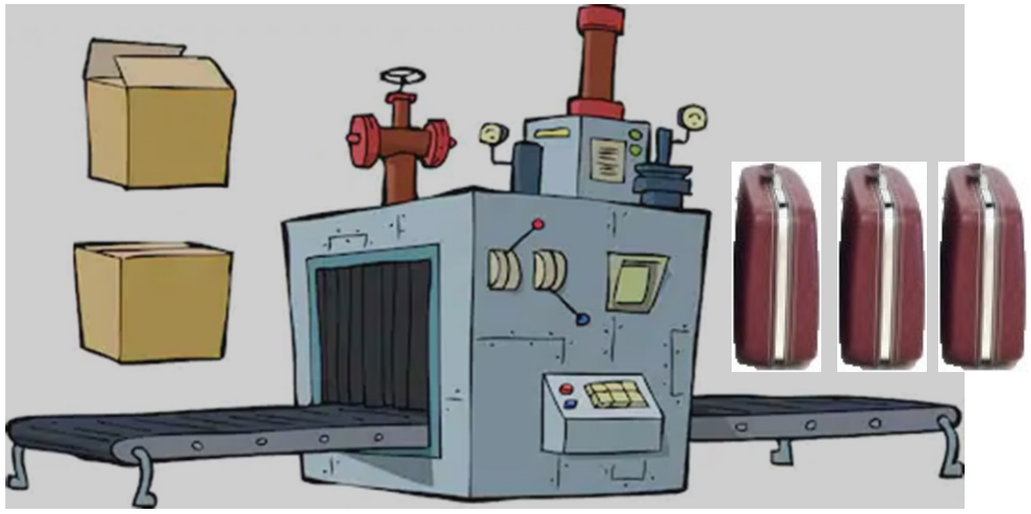
```
function getColdCallList(contacts) {
  let phoneNumbers = [];
  for (let i = 0; i < contacts.length; i++) {
    phoneNumbers[i] = contacts[i].phone;
  }
  return phoneNumbers;
}
```

Array functions

4. Item Transformation

"Apply" a smaller transform, across all elements.

input array



array returned

… of something different!

Does not say anything about sizes of the input/output arrays. The cold-call example happened to be 1:1. But filtering or expanding functions are common too.

An object+function idiom

5. Parameter Mutation

"Work on" the array/record that was passed in.
This is new. And unlike the others.

Example, replace all the a's with b's.

```
function replaceAll(arr, a, b) {
  for (let i = 0; i < arr.length; i++) {
    if (arr[i] === a) {
      arr[i] = b;
    }
  }
}
```

An object+function idiom

5. Parameter Mutation

Not a global variable

"Work on" the array/record that was passed in.

Current e.g. it's "assign to array elements."

Sometimes it's adding/removing elements.

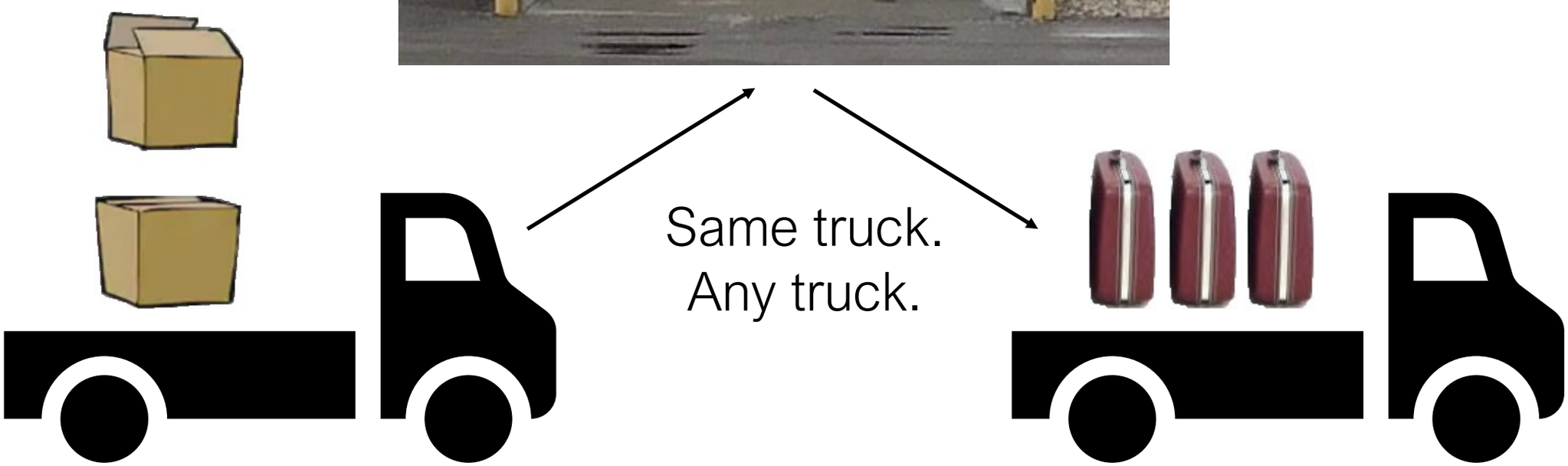
```
function replaceAll(arr, a, b) {
                                    arr[i] = b;
    }
  }
}
```

In this idiom, no concern for how a loop works. There may not be a loop.

No return

An object+function idiom

5. Parameter Mutation



Dot functions

Many built-in array functions use an invocation syntax like this:

indexOf takes two parameters

```
print(arr.indexOf(q));
```

print receives the return of indexOf

- You won't write your functions to get called this way.
- You will have to call some built-in functions this way.
- Treat what's before the dot as an extra parameter.

Which idiom is this?

Which idiom is this?

Built-in array functions

First Group: No mutation, result is returned

```
let a = ['a','b','c','d'];
let b = ['x','y'];

print(a.concat(b));          // a,b,c,d,x,y
print(a.includes('c'));    // true
print(a.indexOf('c'));     // 2
print(shuffle(a, false)); // e.g. b,a,d,c

let startAt = 2;
let stopBefore = 4;
print(a.slice(startAt, stopBefore)); //c,d
```

Built-in array functions

Second Group: With mutation, original is modified

```
let a = ['d','c','b','a'];
let extraItem = 'x';

a.push(extraItem);     print(a); // d,c,b,a,x
a.pop();               print(a); // d,c,b,a
a.unshift(extraItem); print(a); // x,d,c,b,a
a.shift();             print(a); // d,c,b,a

a.sort();              print(a); // a,b,c,d
a.reverse();           print(a); // d,c,b,a
shuffle(a, true);      print(a); // e.g. a,d,c,b
```

Built-in array functions

An extra: With mutation, original is modified

```
let a = ['d','c','b','a'];
let extraItem = 'x';
let midpoint = 2;
```

```
// delete 1 item at midpoint
a.splice(midpoint, 1);
print(a); // d,c,a

// delete 0 items at midpoint and add extraItem
a.splice(midpoint, 0, extraItem);
print(a); // d,c,x,a
```

Strings

In many programming situations, we want to deal with blocks of text.

- Text boxes in a web form
- Text drawn to the screen
- Analyzing text documents for patterns

We need a type to hold blocks of text. JavaScript includes the “String” type to do exactly this.

Literals

To give an explicit string in your program (a *literal*),
put it in quotes.

```
let a = 'hello';
let b = 'world';
let c = ' ';
let d = '*';
let e = '';
let f = 'Lorem ipsum dolor sit amet, elit.';
```

… and any quotes will do.

```
let x = "hello";
print(a===x); // true
```

```
print( "mouse is pressed" );
```

String literals

```
img = loadImage( "data/bird.png" );
```

Literals: special characters

And now the leather-covered sphere came hurtling through the
air,
And Casey stood a-watching it in haughty grandeur there.
Close by the sturdy batsman the ball unheeded sped—
"That ain't my style," said Casey. "Strike one!" the umpire said.

```
let lastLine = ""That ain't my style," said";
```

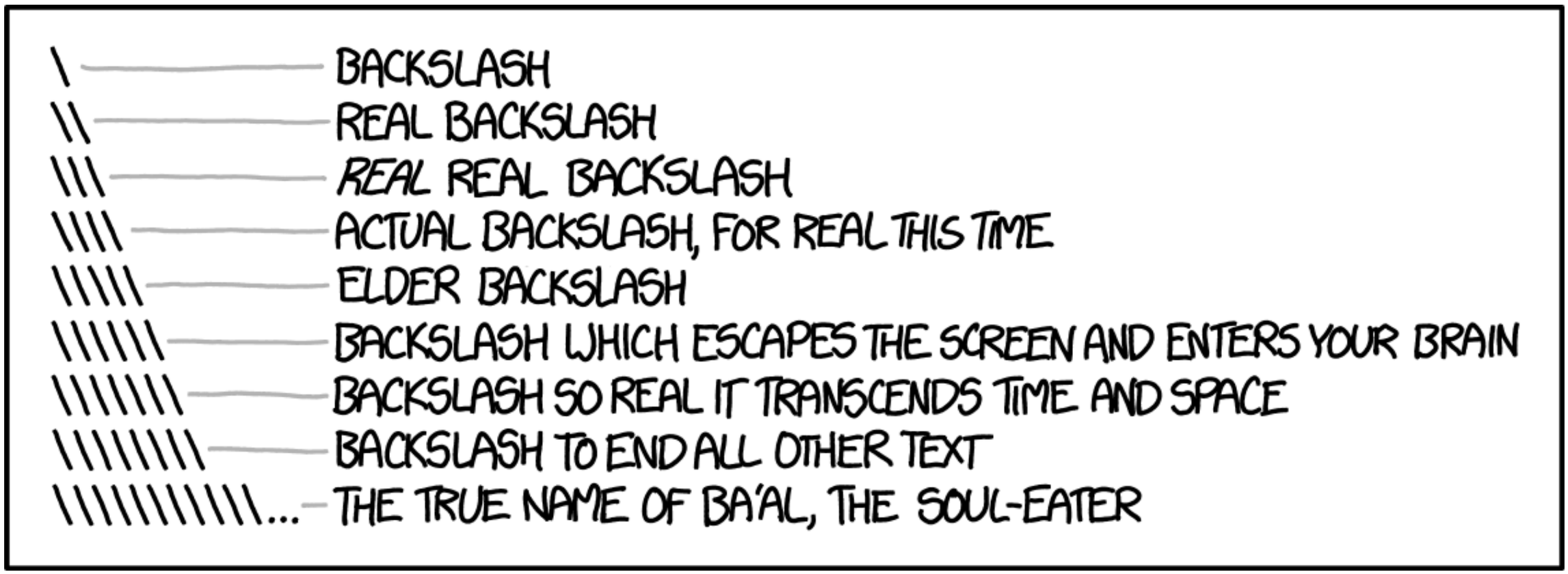
Ernest Lawrence Thayer, *Casey at the Bat* (1888)

Literals: special characters

Use the backslash \ to tell JS about upcoming special characters.

```
let singleqt    = "\'";
let doubleqt    = "\"";
let newline     = "\n";      // like return
let dbldagger1 = "\u2021"; // Unicode num for ‡
let dbldagger2 = "‡";      // often paste is fine

let backslash  = "\\";
```



https://xkcd.com/1638/

Special chars: line breaks

How come? History. Sorry.
Wherefore? Browser compatibility. Sorry.

```
let NL   = "\n";      // preferred, but...
let CR   = "\r";
let CRNL = "\r\n";

print(NL===CR);       // false

let q = ... ;         // is q a line break?
if (q===NL || q===CR || q===CRNL) { ... }
                                  // sorry
```

Literals: + to split long ones

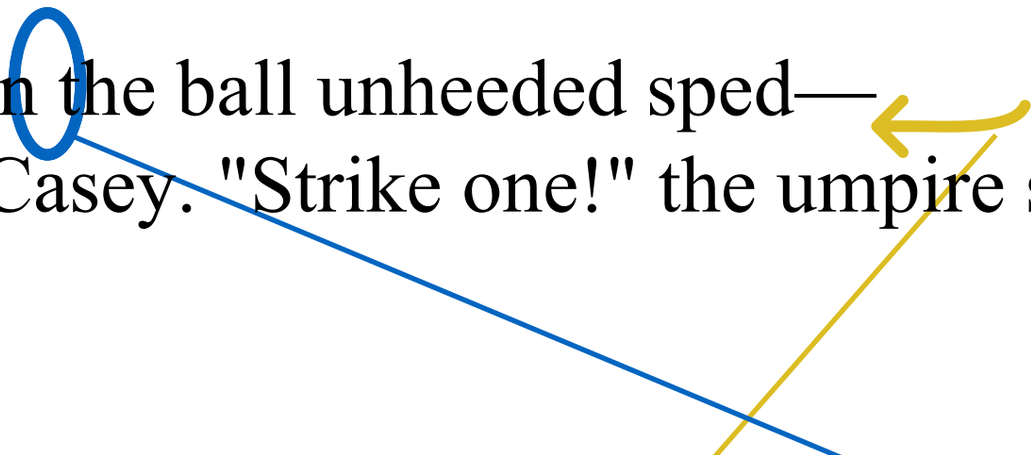
```
one!\" the umpire said.";
```

```
let verse2 = "Close by the sturdy batsman " +
  "the ball unheeded sped—\n" +
  "\"That ain't my style,\" said Casey. " +
  "\"Strike one!\" the umpire said.";

print(verse1===verse2); // true
```

Literals: + to split long ones

Close by the sturdy batsman the ball unheeded sped—
"That ain't my style," said Casey. "Strike one!" the umpire said.



```
let verse2 = "Close by the sturdy batsman " +
  "the ball unheeded sped—\n" +
  "\"That ain't my style,\" said Casey. " +
  "\"Strike one!\" the umpire said.";
```

Literals: pick your quote

```
let abbrev1   = 'didn\'t';
let abbrev2   = "didn't" ;
```

```
let dialogue1 = 'Now hear, "this!"'  ;
let dialogue2 = "Now hear, \"this!\"";

print(abbrev1  ===abbrev2  ); // true
print(dialogue1===dialogue2); // true
```

Strings are just values

```
let str1 = "Hello";
let str2 = str1;


function processString(str,  num) {
   …
}

let str3 = processString(str1, 3.14);


let columns = ["Doric", "Ionic", "Corinthian"];
```

String equality

We often want to compare two strings to see whether they have the same text. They are values, after all!

```
if (str1 === str2) {
   // the strings have the same text
}
```

Concatenation and equality

+ gives LHS then RHS. It concatenates.

Strings are values.

Values don’t care how you got them.

```
let s = "He" ;
print( "Hello"                    );    // Hello
print(          s + "llo"        );    // Hello
print( "Hello" === (s + "llo") );    // true

let n = 2    ;
print( 102                       );    // 102
print(          n + 100          );    // 102
print( 102     === (n + 100)     );    // true
```

Are Strings just, like, Arrays?

Almost, but not quite.
Strings wish they were arrays of characters, but they aren't.
Still, your knowledge of arrays will help you.

```
let wd = ['h','e','l','l','o'];
```

```
let wd = "hello";
```

String vs. Array

Strings are containers. Of *characters.* Almost like arrays.

JS doesn't have a type for characters.
JS doesn't have a type for containers either.

a container
IS ~~*of*~~ *something*
that only exists
in our minds

Dissecting a string gives length-one strings.

```
let s = 'abc';
print(s[1] === 'b'); // true
```

String vs. Array

Strings wish they were arrays of

… well … length-one strings? (Wait, what?)

Even *still*, your knowledge of arrays will help you.

```
let w1 = ['a','b','c'];
let len1 = w1.length;
let char1 = w1[2];
```

```
let w2 = 'abc';
let len2 = w2.length;
let char2 = w2[2];
```

String vs. Array

Strings wish they were arrays of

… well … length-one strings? (Wait, what?)

Even *still*, your knowledge of arrays will help you, to a point.

```
let w1 = ['a','b','c'];
let len1 = w1.length;
let char1 = w1[2];
w1.reverse();
```

```
let w2 = 'abc';
let len2 = w2.length;
let char2 = w2[2];
w2.reverse();
```

```
Uncaught TypeError: w2.reverse is not a function (sketch: line 7)
```

String vs. Array

Strings wish they were arrays of

… well … length-one strings? (Wait, what?)

Even *still*, your knowledge of arrays will help you, to a point.

```
let w1 = ['a','b','c'];
let len1 = w1.length;
let char1 = w1[2];

w1[3] = '!';
print(w1); // a,b,c,!
```

```
let w2 = 'abc';
let len2 = w2.length;
let char2 = w2[2];

w2[3] = '!';
print(w2); // abc
```

String vs. Array

Strings are *immutable*: once you have one, you can't change it. You can assign a different string to the same variable.

```
let w1 = ['a','b','c'];
let len1 = w1.length;
let char1 = w1[2];
w1.reverse();
w1[3] = '!';
```

```
let w2 = 'abc';
let len2 = w2.length;
let char2 = w2[2];
// no
// no
```

str[i] vs. *“a character”*

Sometimes we want a character to be more than a short string.

What do we know about d ?

- It comes before q
- It’s the successor of c
- It does ===`"\u0064"`
- It’s the successor of `"\u0063"`
- It’s the successor of the successor of `"\u0062"`

… some simple arithmetic actually makes sense here!

str[i] vs. *“a character”*

Sometimes we want a character to support simple arithmetic.

```
let deeCode = 'd'.charCodeAt(0);

print( deeCode     ); // 100
print( deeCode - 1 ); //  99
```

```
print( deeCode         === 0x0064 ); // true
print( deeCode - 1     === 0x0063 ); // true
print( deeCode - 1 - 1 === 0x0062 ); // true

// does d come before q?
let qewCode = 'q'.charCodeAt(0);
print( deeCode < qewCode );          // true
```

Some *"character"* info with no codes

What (else) do we know about `d` ?

- It's the lower-case form of `D`

```
print('--d--'.toUpperCase());          // --D--
print('D'     .toLowerCase());          // d
```

- It comes before `q`

```
print('d' < 'q');                      // true
```

The same comparison works on long strings.

Careful that it still acts code-ish when there are capitals.

```
print('aardvark' < 'ant');             // true
print('aardvark' < 'Ant');             // false
print('aardvark'.toUpperCase()
                < 'Ant'.toUpperCase()); // true
```

What does this print?

String vs. Array: using order

```
let a = ['a','b','c'];
let a2 = ['d','e'];

print(a.concat(a2));
// a,b,c,d,e

print(a.slice(1, 3));
// b,c

print(a.indexOf('b'));
// 1
```

```
let s = 'abc';
let s2 = 'de';

print(s + s2 );
// abcde

print(s.substring(1, 3));
// bc

print(s.indexOf('b'));
// 1

print('z'.repeat(3));
// zzz
```

String vs. Array: using order

All the “array versions” on the last slide follow patterns 1..4 (data in, data out; no mutation).

There are often no string equivalents of the built-in functions that use array pattern 5 (mutation):

- `splice`: instead, use `indexOf`, `substring`, +, `join`
- `push`, `pop`, `shift`, `unshift`: ditto; special cases of `splice`
- `reverse`: easy enough to make your own
- `sort`: unusual to want inside one piece text
- `fill`: last slide’s `repeat` (pattern 4) fills in nicely

More on Concatenation

The + operator on strings is very flexible.

```
"Call me" + " " + "Ishmael."

"Ours go to " + 11

"The value of PI is " + PI

"A " + true + " or " + false + " question"

let x, y;
"The point is at (" + x + ", " + y + ")"
```

String and Array: better together

```
let a = ['a','b','c'];
let s = 'abc';

// join : array -> string
print( a.join('-THEN-')  );  // a-THEN-b-THEN-c
print( a.join('')        );  // abc
print( a.join('')  === s );  // true
// split : string -> array
print( 'a-THEN-b-THEN-c'.split('-THEN-') );
                             // a,b,c
print( s.split('')       );  // a,b,c
print( s.split('') === a );  // false
```

strings are values

arrays are objects

What does this print?

What does this print?

What does this print?

Parsing strings

We often obtain “raw text” from external sources, and need to *parse* it into meaningful data.

The built-in functions int() and float() work on strings and arrays of strings.

```
let a = int("1234");
let b = float("567.89");

let strs = ["-81", "0", "36"];
let arr = int(strs);
```

Outputting text

The p5 print() function will write any text (or really, any value at all) to the console. Handy for debugging!

The built-in text() function will draw text at a given position in the sketch window, using the current fill colour.

See also textSize(), textFont(), textAlign().

Working with Text/Strings

Specify a specific Font

```
textFont("Georgia");
```

Load in your favourite Font!

```
let myFont = loadFont("assets/inconsolata.otf");
textFont(myFont);
```

Working with Text/Strings

Specify the fill color

```
fill(0);
```

Specify the size

```
textSize(25);
```

Specify how to align the text

```
textAlign(CENTER, BOTTOM);
```

Display the text

```
let txt = "my text";
text(txt, 10, 10);
```

```
function setup() {
  createCanvas(275, 400);

  textSize(72);
  colorMode(HSB, 255);
  background(0, 0, 255);
  for (let y = 80; y < 380; y += 15) {
    fill(map(y, 80, 380, 0, 255), 255, 255);
    text("CS 106" , 10, y);
  }
}
```