Arrays and Strings
Arrays are examples of:

JavaScript p5 Objects

JavaScript p5 knows they can get big.
Copying big things can make our program slow.
What does this print?

```plaintext
let a = 1;
let b = 2;
a = b;
b = 3;
print(a + b);
```

(A) 3
(B) 4
(C) 5
(D) 6
(E) 7
What does this print?

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

(A) 3  (B) 4  (C) 5  (D) 6  (E) 7
Arrays are just values...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*.

```javascript
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]);
function setup() {

    let a = [1, 2, 3];

    let b = [4, 5, 6, 7];

    a = b;

    b[0] = 9;

    print("a: " + a[0] + " B: " + b[0]);

    print("Array a is: " + a);

    print("Array b is: " + b);

}
References

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

Arrays are passed around by reference (arrows).
Comparing Arrays: be careful

let arr1 = [1, 2, 3];
let arr2 = [1, 2, 3];
print(arr1 === arr2); // false

• JavaScript p5 compares the arrows
• The arrows point to different places
• So JavaScript p5 determines that the arrays are not equal
Comparing Arrays: be careful

```javascript
let arr1 = [1, 2, 3];
let arr2 = arr1;
print(arr1 === arr2); // true
```

- JavaScript p5 compares the arrows
- The arrows point at the same place
- So JavaScript determines that the arrays are equal
Comparing Arrays: be careful

```javascript
let arr1 = [1, 2, 3];
let arr2 = [1, 2, 3];
let eql = true;
for (let i = 0; i < arr1.length; i++) {
    if (arr1[i] != arr2[i]) {
        eql = false;
    }
}
print(eql); // true
```

• One solution: Use a loop to check each array element.
What does this print?

```javascript
let arr1 = [1, 2, 3, 4];
let arr2 = [];
arr2[0] = arr1[2];
print(arr1);
```

A) [1, 2, 3, 4]
B) 2
C) [2, 1, 2, 3, 4]
D) [1, 1, 2, 3, 4]
E) Error
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.

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

We can use more than one at once. We should know them, recognize them.
Array idioms

1. Item Consumption

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

```javascript
// 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.

```javascript
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

```javascript
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.

```javascript
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].

```javascript
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).

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

4. Parameter Mutation

“Work on” the array that was passed in. This is new. And unlike the others.

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

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

- We have seen dot notation syntax before with .length
- Here we show “.indexOf()”

```javascript
function setup() {
  let q = "c";
  let arr = ['a', 'b', 'c', 'd'];
  let result = arr.indexOf(q);
  print(result);
}
```
Built-in array functions

First Group: No mutation, result is returned

```javascript
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

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

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

```javascript
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");

img = loadImage("bird.png");
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;
Literals: special characters

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

```javascript
let singleqt = "\"";
let doubleqt = "\"\";
let newline = "\n"; // like return
let dbldagger1 = "\u2021"; // Unicode num for ‡
let dbldagger2 = "‡"; // often paste is fine
let backslash  = "\"\";
```
\ --- \ BACKSLASH
\\ --- \ REAL BACKSLASH
\\\\ --- \ REAL REAL BACKSLASH
\\\\\ --- \ ACTUAL BACKSLASH, FOR REAL THIS TIME
\\\\\\ --- \ ELDER BACKSLASH
\\\\\\\ --- \ BACKSLASH WHICH ESCAPES THE SCREEN AND ENTERS YOUR BRAIN
\\\\\\\\ --- \ BACKSLASH SO REAL IT TRANSCENDS TIME AND SPACE
\\\\\\\\\ --- \ BACKSLASH TO END ALL OTHER TEXT
\\\\\\\\\\ --- \ THE TRUE NAME OF BA’AL, THE SOUL-EATER
Literals: pick your quote

let abbrev1 = 'dint\'t';
let abbrev2 = "dint" ;

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

print(abbrev1 === abbrev2 ); // true
print(dialogue1=== dialogue2); // true
Strings are just values

```javascript
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!

```java
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.

```plaintext
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.

```javascript
let wd = ['h','e','l','l','o'];
let wd = "hello";
```
String vs. Array

Strings are almost like arrays.

JS doesn’t have a type for characters.

Dissecting a string gives length-one strings.

```javascript
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.

```javascript
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();

🛑 p5.js says: There's an error as "reverse" could not be called as a function
```
An Array of Strings

```javascript
let w1 = ['a', 'b', 'c'];

let len1 = w1.length;

let char1 = w1[2];

print(w1);
print(len1);
print(char1);

w1.reverse();
print(w1);
```

```
Array ["a","b","c"]
3
c
Array ["c","b","a"]
```
Strings

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

print(w2);
print(len2);
print(char2);

w2.reverse();
print(w2);

p5.js says: There's an error as "reverse" could not be called as a function
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.

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

w1[3] = 'd';

print(w1); // a,b,c,d
```

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

w2[3] = 'd';

print(w2); // abc
```
Arrays vs Strings

let w1 = ['a', 'b', 'c'];

w1[3] = 'd';

print(w1); // a, b, c, d

let w2 = 'abc';

w2[3] = 'd';

print(w2); // abc

Array ["a", "b", "c", "d"]

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

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

let w2 = 'abc';
let len2 = w2.length;
let char2 = w2[2];
// yes
// yes
```
More dot notation on Strings

• Lower-case and upper-case
  print('--d--'.toUpperCase()); // --D--
  print('D'.toLowerCase()); // d
  print('d' === 'D'.toLowerCase()); // true

The same comparison works on long strings. Convert everything to lower-case or upper-case:

print('aardvark' < 'ant'); // true
print('aardvark' < 'Ant'); // false
print('aardvark'.toUpperCase()
  < 'Ant'.toUpperCase()); // true
What does this print?

```javascript
let s = ["CBC", "Global", "CBC", "CTV"];

let result1 = (s[0] === s[2] );
let result2 = (s[0][0] === s[3][0]);

print(result1 + " " + result2);
```

A) true true  
B) true false  
C) false true  
D) false false  
E) (an error)
String vs. Array

```javascript
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
```
More on Concatenation

The + operator on strings is very flexible.

```plaintext
let a = "Call me" + " " + "Ishmael.";
let b = "Ours go to " + 11;
let c = "The value of PI is " + PI;
let d = "A " + true + " or " + false + " question";
let x = 1.5;
let y = 2.5;
let e = "The point is at (" + x + ", " + y + ")";

print(a, b, c, d, e);
```

```
Call me Ishmael.
Ours go to 11
The value of PI is 3.141592653589793
A true or false question
The point is at (1.5, 2.5)
```
String and Array: better together
Join and Split are the workhorses of most practical text processing.

```javascript
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
```
What does this print?

```
let d = 39;
print("abc" + d);
```

(A) abc
(B) abcd
(C) "abc"d
(D) abc39
(E) abcd39
What does this print?

print("123" + 456);

(A) 123
(B) 123456
(C) 579
(D) 12459
(E) Nothing, it’s an error
What does this print?

```javascript
function setup() {
let s = ["ABC", "NBC", "CBC", "CTV"]; 
let result; 
result = s[1] + ":" + s[2]; 
print(result);
}
```

A) ABC:NBC
B) NBC:CBC
C) CBC:CTV
D) ABCNBC
E) NBCCBC
Parsing strings

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

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

```plaintext
let a = int("1234");
let b = float("567.89");
let strs = ["-81", "0", "36"];
let arr = int(strs);
print(a, b, strs, arr);
```

1234
567.89
2x \textbf{Array} ["-81","0","36"]
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
```javascript
textFont("Georgia");
```

You can use any WebSafeFont (Google to see list).

Load in your favourite Font in preload()
```javascript
let myFont = loadFont("assets/inconsolata.otf");
```

```javascript
textFont(myFont);
```
Best Web Safe Fonts for HTML and CSS
The following list are the best web safe fonts for HTML and CSS:

- Arial (sans-serif)
- Verdana (sans-serif)
- Helvetica (sans-serif)
- Tahoma (sans-serif)
- Trebuchet MS (sans-serif)
- Times New Roman (serif)
- Georgia (serif)
- Garamond (serif)
- Courier New (monospace)
- Brush Script MT (cursive)

Credit: w3Schools.com
let index = 0;

let myFonts = [
    "Arial",
    "Verdana",
    "Helvetica",
    "Tahoma",
    "Trebuchet MS",
    "Times New Roman",
    "Georgia",
    "Garamond",
    "Courier New",
    "Brush Script MT"
];
function setup() {
    createCanvas(600, 400);
    background(220);
    textAlign(CENTER, CENTER);
    textFont(myFonts[index]);
    textSize(25);
}

function draw() {
    background(220);
    text("This is font: " + myFonts[index], 25, height / 2);
    text("1 2 3 4 5 6 7 8 9 0", 25, height / 2 + 50);
}

function keyPressed() {
    index = index + 1;
    if (index >= myFonts.length) {
        index = 0;
    }
    textFont(myFonts[index]);
}
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);
    }
}