Data 4: Arrays

This unit introduces arrays of data.

Syntax introduced:
`Array, [] (array access), new, Array.length`
`append(), shorten(), expand(), arraycopy()`

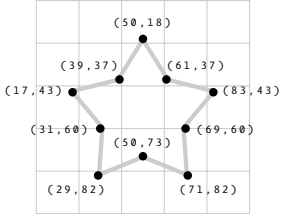
The term array refers to a structured grouping or an imposing number—"The dinner buffet offers an array of choices," "The city of Los Angeles faces an array of problems." In computer programming, as array is a set of data elements stored under the same name. Arrays can be created to hold any type of data, and each element can be individually assigned and read. There can be arrays of numbers, characters, sentences, boolean values, etc. Arrays might store vertex data for complex shapes, recent keystrokes from the keyboard, or data read from a file.

Five integer variables (1919, 1940, 1975, 1976, 1990) can be stored in one integer array rather than defining five separate variables. For example, let's call this array "dates" and store the values in sequence:

dates	1919	1940	1975	1976	1990
	[0]	[1]	[2]	[3]	[4]

Array elements are numbered starting with zero, which may seem confusing at first but is important for more advanced programming. The first element is at position [0], the second is at [1], and so on. The position of each element is determined by its offset from the start of the array. The first element is at position [0] because it has no offset; the second element is at position [1] because it is offset one place from the beginning. The last position in the array is calculated by subtracting 1 from the array length. In this example, the last element is at position [4] because there are five elements in the array.

Arrays can make the task of programming much easier. While it's not necessary to use them, they can be valuable structures for managing data. Let's begin with a set of data points we want to add to our program in order to draw a star:



The following example to draw this shape demonstrates some of the benefits of using arrays, for instance, avoiding the cumbersome chore of storing data points in

individual variables. The star has 10 vertex points, each with 2 values, for a total of 20 data elements. Inputting this data into a program requires either creating 20 variables or using an array. The code (below) on the left demonstrates using separate variables. The code in the middle uses 10 arrays, one for each point of the shape. This use of arrays improves the situation, but we can do better. The code on the right shows the data elements can be grouped logically together in two arrays, one for the x-coordinates and one for the y-coordinates.

Separate variables

```
int x0 = 50;
int y0 = 18;
int x1 = 61;
int y1 = 37;
int x2 = 83;
int y2 = 43;
int x3 = 69;
int y3 = 60;
int x4 = 71;
int y4 = 82;
int x5 = 50;
int y5 = 73;
int x6 = 29;
int y6 = 82;
int x7 = 31;
int y7 = 60;
int x8 = 17;
int y8 = 43;
int x9 = 39;
int y9 = 37;
```

One array for each point

```
int[] p0 = { 50, 18 };
int[] p1 = { 61, 37 };
int[] p2 = { 83, 43 };
int[] p3 = { 69, 60 };
int[] p4 = { 71, 82 };
int[] p5 = { 50, 73 };
int[] p6 = { 29, 82 };
int[] p7 = { 31, 60 };
int[] p8 = { 17, 43 };
int[] p9 = { 39, 37 };
```

One array for each axis

```
int[] x = { 50, 61, 83, 69, 71,
            50, 29, 31, 17, 39 };
int[] y = { 18, 37, 43, 60, 82,
            73, 82, 60, 43, 37 };
```

This example shows how to use the arrays within a program. The data for each vertex is accessed in sequence with a `for` structure. The syntax and usage of arrays is discussed in more detail in the following pages.

```
int[] x = { 50, 61, 83, 69, 71, 50, 29, 31, 17, 39 };
int[] y = { 18, 37, 43, 60, 82, 73, 82, 60, 43, 37 };

beginShape();
// Reads one array element every time through the for()
for (int i = 0; i < x.length; i++) {
  vertex(x[i], y[i]);
}
endShape(CLOSE);
```

33-01

Using arrays

Arrays are declared similarly to other data types, but they are distinguished with brackets, [and]. When an array is declared, the type of data it stores must be specified. After the array is declared the array must be created with the keyword "new." This additional step allocates space in the computer's memory to store the array's data. After the array is created, the values can be assigned. There are different ways to declare, create, and assign arrays. In the following examples explaining these differences, an array with five elements is created and filled with the values 19, 40, 75, 75, and 90. Note the different way each method for creating and assigning elements of the array relates to `setup()`.

33-02

```
int[] data;               // Declare
void setup() {
  size(100, 100);
  data = new int[5];      // Create
  data[0] = 19;           // Assign
  data[1] = 40;
  data[2] = 75;
  data[3] = 76;
  data[4] = 90;
}
```

33-03

```
int[] data = new int[5];  // Declare, create
void setup() {
  size(100, 100);
  data[0] = 19;           // Assign
  data[1] = 40;
  data[2] = 75;
  data[3] = 76;
  data[4] = 90;
}
```

33-04

```
int[] data = { 19, 40, 75, 76, 90 };  // Declare, create, assign
void setup() {
  size(100, 100);
}
```

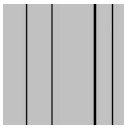
The previous three examples assume the arrays are used in a sketch with `setup()` and `draw()`. If arrays are not used with these functions, they can be created and assigned in the simpler ways shown in the following examples.

```
int[] data;         // Declare                                    33-05
data = new int[5];  // Create
data[0] = 19;       // Assign
data[1] = 40;
data[2] = 75;
data[3] = 76;
data[4] = 90;
```

```
int[] data = new int[5];  // Declare, create                      33-06
data[0] = 19;             // Assign
data[1] = 40;
data[2] = 75;
data[3] = 76;
data[4] = 90;
```

```
int[] data = { 19, 40, 75, 76, 90 };  // Declare, create, assign  33-07
```

The declare, create, and assign steps allow an array's values to be read. An array element is accessed using the name of the variable followed by brackets around the position from which you are trying to read.



```
int[] data = { 19, 40, 75, 76, 90 };                             33-08
line(data[0], 0, data[0], 100);
line(data[1], 0, data[1], 100);
line(data[2], 0, data[2], 100);
line(data[3], 0, data[3], 100);
line(data[4], 0, data[4], 100);
```

Remember, the first element in the array is in the 0 position. If you try to access a member of the array that lies outside the array boundaries, your program will terminate and give an `ArrayIndexOutOfBoundsException`.

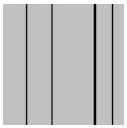
```
int[] data = { 19, 40, 75, 76, 90 };                             33-09
println(data[0]);  // Prints 19 to the console
println(data[2]);  // Prints 75 to the console
println(data[5]);  // ERROR! The last element of the array is 4
```

The `length` field stores the number of elements in an array. This field is stored within the array and can be accessed with the dot operator (p. 107). The following example demonstrates how to utilize it.

```
int[] data1 = { 19, 40, 75, 76, 90 };
int[] data2 = { 19, 40 };
int[] data3 = new int[127];
println(data1.length); // Prints "5" to the console
println(data2.length); // Prints "2" to the console
println(data3.length); // Prints "127" to the console
```

33-10

Usually, a `for` structure is used to access array elements, especially with large arrays. The following example draws the same lines as the previous ones but uses a `for` structure to iterate through every value in an array.



```
int[] data = { 19, 40, 75, 76, 90 };
for (int i = 0; i < data.length; i++) {
  line(data[i], 0, data[i], 100);
}
```

33-11

A `for` structure can also be used to put data inside an array—for instance, it can calculate a series of numbers and then assign each value to an array element. The following example stores the values from the `sin()` function in an array within `setup()` and then displays these values as the stroke values for lines within `draw()`.

```
float[] sineWave = new float[width];

for (int i = 0; i < width; i++) {
  // Fill the array with values from sin()
  float r = map(i, 0, width, 0, TWO_PI);
  sineWave[i] = abs(sin(r));
}

for (int i = 0; i < sineWave.length; i++) {
  // Set the stroke values to numbers read from the array
  stroke(sineWave[i] * 255);
  line(i, 0, i, height);
}
```

33-12

Storing the coordinates of many elements is another way to use arrays to make a program easier to read and manage. In the following example, the `x[]` array stores the x-coordinate for each of the 12 elements in the array, and the `speed[]` array stores a rate corresponding to each. Writing this program without arrays would have required 24 separate variables. Instead, it's written in a flexible way; simply changing the value assigned to `numLines` sets the number of elements drawn to the screen.