Chapter 4: Data Structures: Objects and Arrays

Objects allow us to group values—including other objects—together and thus build more complex structures.

Data sets

array - a data type specifically for storing sequences of values

```
var listOfNumbers = [2, 3, 5, 7, 11]; console.log(listOfNumbers[1]);
// → 3
console.log(listOfNumbers[1 - 1]);
// → 2
```

Properties

Almost all JavaScript values have properties. The exceptions are null and undefined. If you try to access a property on one of these nonvalues, you get an error.

ways to access properties:

. dot eg. null.x — part after dot must be a valid variable name, and it directly names the property. fetches the property of value named “x”

[ ] square brackets —eg. value[x] — the expression between the brackets is evaluated to get the property name

Methods

property examples: length, toUpperCase, toLowerCase, push, pop, join

Properties that contain functions are generally called **methods** of the value they belong to. As in, “toUpperCase is a method of a string”.

Objects

Values of the type **object** are arbitrary collections of properties, and we can add or remove these properties as we please.

Each property is written as a name, followed by a colon, followed by an expression that provides a value for the property.

```
var day1 = {
```

```
        squirrel: false,
        events: ["work", "touched tree", "pizza", "running", "television"]
};
console.log(day1.squirrel);
// → false
console.log(day1.wolf);
// → undefined
day1.wolf = false;
console.log(day1.wolf);
// → false
```

- Properties whose names are not valid variable names or valid numbers have to be quoted.

```
var descriptions = {
        work: "Went to work",
        "touched tree": "Touched a tree"
};
```

This means that curly braces have two meanings in JavaScript. At the start of a statement, they start a block of statements. In any other position, they describe an object.

- Reading a property that doesn't exist will produce the value undefined,

- It is possible to assign a value to a property expression with the = operator. This will replace the property's value if it already existed or create a new property on the object if it didn't.

The **delete** operator cuts o a tentacle from such an octopus. It is a unary operator that, when applied to a property access expression, will remove the named property from the object.

```
var anObject = {left: 1, right: 2}; console.log(anObject.left);
// → 1
delete anObject.left; console.log(anObject.left);
// → undefined
```

The binary **in** operator, when applied to a string and an object, returns a Boolean value that indicates whether that object has that property.

setting property to undefined = still has property (with uninteresting value) vs. deleting property = property no longer present & will return false

Arrays, then, are just a kind of object specialized for storing sequences of things. If you

evaluate typeof [1, 2], this produces "object".

Mutability

object values *can* be modified vs. eg. numbers, strings, and Booleans - are all **immutable -** it is impossible to change an existing value of those types.

- you can combine and derive new values from strings, but when you take a specific string value, that value will always remain the same - t he text inside it cannot be changed

JavaScript's == operator, when comparing objects, will return true only if both objects are *precisely* the same value. Comparing different objects will return false, even if they have identical contents. There is no "deep" comparison operation built into JavaScript.

The lycanthrope's log

Computing correlation

A **map** is a way to go from values in one domain (in this case, event names) to corresponding values in another domain (in this case, ϕ coef- cients).

Further arrayology

We saw **push** and **pop**, which add and remove elements at the end of an array, earlier in this chapter. The corresponding methods for adding and removing things at the start of an array are called **unshift** and **shift**.

The **indexOf** method has a sibling called **lastIndexOf**, which starts search- ing for the given element at the end of the array instead of the front. Both **indexOf** and **lastIndexOf** take an optional second argument that in- dicates where to start searching from.

slice takes a start index and an end index and returns an array that has only the elements between those indices

- Strings also have a slice method, which has a similar effect.

The **concat** method can be used to glue arrays together, similar to what the + operator does for strings.

Strings and their properties

string are immutable (cannot be changed)

slice, **indexOf**

string's indexOf can take a string containing more than one character vs. array method looks only for a single method

The **trim** method removes whitespace (spaces, newlines, tabs, and similar characters) from the start and end of a string.

length, **charAt**

var string = "abc"; console.log(string.length);
// → 3 console.log(string.charAt(0)); // → a
console.log(string[1]);
// → b

The arguments object

Whenever a function is called, a special variable named **arguments** is added to the environment in which the function body runs. This variable refers to an object that holds all of the arguments passed to the function.

the arguments object has:

- a length property that tells us the number of properties really passed to the function.
- a property for each argument, named 0, 1, 2, etc.
- similar to an array, but doesn't have array methods (like **slice** or **indexOf**)

The Math object

grab-bag of number-related utility functions:

Math.max, Math.min, Math.sqrt, cos (cosine), **sin** (sine), **tan** (tangent), **acos**, **asin**, **atan**, **Math.PI, Math.random, Math.floor** (rounds down to the nearest whole number), **Math.ceil** (for "ceiling", which rounds up to a whole number) and **Math.round** (to the nearest whole number).

The **Math** object is used simply as a container to group a bunch of related functionality. It provides a *namespace* so that all these functions don't have to be global variables.

Many languages will stop you, or at least warn you, when you are defining a variable with a name that is already taken. JavaScript does neither, so be careful.

The global object

The global scope, the space in which global variables live, can also be approached as an object in JavaScript. Each global variable is present as a property of this object. In browsers, the global scope object is stored in the **window** variable.