JavaScript Basics

- At this point, you should have reached a certain comfort level with typing and running JavaScript code — assuming, of course, that someone has already written it for you
- This handout aims to summarize some basic building blocks that will get you on the road to writing some of your own JavaScript programs
- If you're interested more details, either consult me, the recommended textbook, or the web — whatever you feel will work best for you **:)**

The Big Picture

- JavaScript (and many other programming languages) have the following basic building blocks (there are more, of course — but this is what we have for now):

  Expressions *Variables* *Statements*

- Of these building blocks, the *statement* is the construct that plays the closest role to a plain English instruction — a JavaScript program is essentially a *sequence of statements*, each of which may involve one or more *expressions* or *variables* to get its work done

Expressions

- Expressions are the fundamental "things" or "nouns" in JavaScript — they are pieces of code that are *evaluated* to determine the "thing" that this code represents
- Examples of such expressions (and their corresponding "things" or computed *values*) include:

`2`	the number 2
`"hello world"`	the phrase "hello world"
`(2 + 8.1) * 5`	the number 50.5
`9 > 4`	something that is true
`"dog" + "house"`	the word "doghouse"
`"bad" === "good"`	something that is false

- Expressions consist of *values* and *operators*
- Every value in JavaScript is either a *Boolean* (*true* or *false*), a number, a *string* (i.e., a piece of text, including letters, numbers, punctuation, or other symbols), a special value called *undefined*, a special value called *null*, or an *object*
- Of these, Booleans, numbers, and strings are easiest values to understand, because we probably already use them a lot in daily life
- *Operators* represent actions that combine or manipulate values to produce a new value — for example, multiplication operator (represented by `*` in JavaScript) combines two numbers and to compute their product

Boolean Values

There are ultimately only two Boolean values: *true* and *false* (sorry, no "maybe"s here) — Booleans are most useful not in these forms (though JavaScript *does* understand them), but as the results of operators:

Operator	Meaning	Sample Expression	Value
`===`	equal to	`7 === 5`	`false`
`!==`	not equal to	`"dog" !== "cat"`	`true`
`<`	less than	`10 < 100`	`true`
`>`	greater than	`10 > 100`	`false`
`<=`	less than or equal to	`5 <= 0`	`false`
`>=`	greater than or equal to	`12 >= 12`	`true`

- There are also operators that combine or manipulate Boolean expressions themselves: `&&` ("and"), `||` ("or"), `^` ("exclusive or"), and `!` ("not")

`x`	`y`	`x && y`	`x || y`	`x ^ y`	`!x`
`true`	`true`	`true`	`true`	`false`	`false`
`true`	`false`	`false`	`true`	`true`	`false`
`false`	`true`	`false`	`true`	`true`	`true`
`false`	`false`	`false`	`false`	`false`	`true`

- Combined with the examples above, you can get expressions like: `!(7 === 5)` (*false*), `(10 < 100) || (5 <= 0)` (*true*), `("dog" !== "cat") && ("cat" !== "mouse")` (*true*)
- Note the use of parentheses to "group" parts of the expression together

N.B. Some sources use == for "equals" and != for "not equals" — we prefer === and !== because these provide "stricter" interpretations of equality and inequality.

Numbers

- Number expressions very closely resemble familiar, handwritten arithmetic, with a few wrinkles:
  - Huge numbers can be written using "scientific notation," roughly interpreted as "the number before *E* (or e) times 10 raised to the number after *E* (or e)" — `3.6288e6` is 3,628,800; `5.390E-44` is 5.390×10^{-44}
  - The operators `+` (addition) and `-` (subtraction) are what you'd expect; there is also `*` (multiplication), `/` (division), and `%` (modulo, or remainder: `18 % 5 === 3`)

- Other operators are available — note "`Math. ...`" as a common prefix: `Math.floor(2.8)` is 2; `Math.sqrt(16)` is 4; `Math.pow(2.5, 4)` is 39.0625
- As in most programming languages, there is such a thing as a largest and smallest value that JavaScript can handle — any values beyond them yield the special values `Infinity` and `-Infinity`
- Another special value is `NaN` ("not a number"), which JavaScript computes when you give it an expression that, uh, is not a number (e.g., `0/0`, `"dog" - "cat"`, `Infinity - Infinity`, `NaN + 42`, etc.)
- *Precision*, or "how exact" a numerical expression is, also has limits: try the one-liner `alert(0.1 + 0.2);`

Strings

- Values that we typically think of as text, words, or phrases fall under the technical term *string* — symbols (or *characters*) that are *strung* together
- The notion of a "symbol" here is actually quite broad: it adheres to a standard called *Unicode* and encompasses way more than the alphabet, numbers, and punctuation
- String values are written within double quotes (e.g., `"string"`) or single quotes (e.g., `'string'`), all on *one line*
- Special symbols are preceded by a backslash (\) — ask me if you're curious about these

- There are dozens of string operations…to name a few:

Operator	Sample Expression	Value	
`length`	`"Hello, human".length`	`12`	*number*
`indexOf`	`"Where".indexOf("here")`	`1`	*number*
`toLowerCase`	`"Shrink ME!".toLowerCase()`	`"shrink me!"`	*string*
`toUpperCase`	`"Rise, Vader".toUpperCase()`	`"RISE, VADER"`	*string*
`replace`	`"boo".replace("oo", "ird")`	`"bird"`	*string*
`charAt`	`"You're my BFF".charAt(3)`	`"'"`	*string*

- When a user provides information using `prompt`, the resulting values are always strings — you need special operations such as `parseInt` and `parseFloat` to turn them into numbers (i.e., `"2"` is not the same as `2`)
- The + operator is "overloaded" — with numbers, it does addition, while with one or more strings, it connects strings together (*concatenation*)

Variables

- Sometimes you want to *store* or *save* the value of an expression for later use or manipulation
- This storage mechanism is called a *variable* — it holds a value, and has a name (so you can refer to it)
- To "create" a variable, you *declare* it: `var answer;`
- To give it a value, you *assign* an expression to it anytime after it has been declared: `answer = 21 * 2;`
- You may declare and assign a variable in a single bound: `var answer = 42;`

- The best part about variables is that you can use them in expressions — note the following program:

```
var x = 2;        // Declares x, initializing it to 2.
alert(x);         // Alerts 2.
alert(10 * x);    // Alerts 20.
var y;            // Declares y without an explicit initial value.
alert(y);         // Alerts undefined.
y = x * 5;        // Assigns 10 to y, because x is still 2.
var z = y;        // Declares z, initializes it to 10.
y = "dog";        // Assigns "dog" to y, overwriting the old value 10.
alert(y + z);     // Alerts "dog10", because z is still 10.
```

- As an aside, observe that, although the program seems to not do anything useful, it actually does: it shows you how to use variables in expressions…you can say that usefulness is in the eye of the beholder **:)**
- Using a [non-existent] variable before declaring it results in an error (browsers vary on how this is reported)

N.B. JavaScript does allow assignment without declaration (e.g., `title = "Twilight";`), but this is considered to be a language flaw. So, always use `var` when declaring variables.

Arrays

- We take a moment to mention a special kind of value in JavaScript (and other programming languages): an *array*
- Arrays are *sequences of values*: if `a` is a variable to which an array has been assigned, `a[0]` represents its first value, `a[1]` represents its second value, and so on
- Arrays are written in between square brackets (`[ ]`), with individual values separated by commas (`,`):

```
var fib = [0, 1, 1, 2, 3, 5, 8];
var words = ["how", "now", "brown", "cow"];
var arrays = [0, 1, ["array", "in", "an", "array"], 5, "wow"];
```

- For an array `a`, the expression `a.length` yields the number of elements in `a`
- Add values to an array using `push` (to add to the end) or `unshift` (to add to the beginning)
- Remove values from an array using `pop` (to remove from the end) or `shift` (to remove from the beginning)
- You can even `sort` an array — but, by default, this treats all values like strings, so that `10` will be placed before `2`

```
var a = [];           // a is an array of length 0.
var b = [3, 5];       // b has length 2.
b.push(2);            // Now b is [3, 5, 2].
b.unshift(7);         // Now b is [7, 3, 5, 2].
a.push(3, 10, 5);     // Now a is [3, 10, 5].
alert(a.pop());       // Alerts 5 and changes a to [3, 10].
alert(a.shift());     // Alerts 3 and changes a to [10].
b.push(a[0], 1);      // b is now [7, 3, 5, 2, 10, 1].
b.sort();             // b is now [1, 10, 2, 3, 5, 7].
```

Statements

- We come full circle with *statements* — as mentioned, a JavaScript program is essentially a *sequence of statements*
- Statements are *executed* when the program is run
- We have mentioned before that semicolons (;) end statements; the full rule is that that every statement ends with a semicolon unless it already ends with a right curly brace (})
- *Declaration* and *assignment* are simple types of statements (note how they ended with semicolons)

- *Conditional* statements do different actions depending on some condition: they consist of an `if` part, zero or more `else if` parts, and an optional `else` part
- The `if` and `else if` parts include a Boolean expression, enclosed between parentheses (`( )`) — the truth of this expression determines what actions are taken
- All parts provide a sequence of statements enclosed between curly braces (`{ }`) and indented for readability:

```
if (score >= 90) {          // The score variable may have either been
    grade = "A";            // assigned to directly, or provided by the
} else if (score >= 80) {   // user in response to a prompt instruction.
    grade = "B";
} else if (score >= 70) {   // The grade variable is assumed to have been
    grade = "C";            // declared prior to reaching this code.
} else {
    grade = "F";
}
alert("The letter grade for " + score + " is " + grade + ".");
```

Loops

- *Loop* statements execute a set of statements over and over again — this activity, called *iteration*, is a key concept in many programming languages
- The *while* statement performs statements repeatedly as long as a given condition evaluates to true — it starts with the keyword `while`, followed by the condition in parentheses, followed by the statements to repeat, indented and between curly braces
- The *for* statement also loops as long as a condition is true, but allows for some code to run at certain times

- The program below keeps asking for a guess until the user gets it right:

```
// Get a random number between 0 and 25, inclusive.
var index = Math.floor(Math.random() * 26);

// Get a random letter.
var letter = "ABCDEFGHIJKLMNOPQRSTUVWXYZ".charAt(index);

var numberOfTries = 1;
while (prompt("Enter a guess for my letter:") !== letter) {
    numberOfTries = numberOfTries + 1;
}
alert("You guessed it in " + numberOfTries + " tries.");
```

- This one gathers up the first letter of each word in the `words` array:

```
// Alerts a string made up of the initial characters of each array item.
var words = ["Rats", "are", "very", "intellegent"];  // Get it?  :)
var result = "";
for (var i = 0; i < words.length; i++) {
    result = result + words[i].charAt(0);
}
alert(result);
```

Mix and Match

- Expressions, variables, and statements are *building blocks* — their power truly emerges when used [correctly] in combination with each other
- You've already seen how expressions can take a string and produce a number (`length`), or take numbers to produce a Boolean value (`===`, `!==`, `<`, `>`, etc.)
- A simple form of *recursion* exists as well: expressions can contain more expressions (typically *nested* in parentheses `( )`), and statements can contain more statements (loops inside conditionals or vice versa)

Overall Structure

At this early stage, you might want to give your programs the following superstructure:

- Specification of input, whether by `prompt` (with appropriate conversion if necessary) or direct variable declaration and assignment
- The instructions for the algorithm, leading to its answer stored in a variable
- Display of the variable within an appropriate message, typically using `alert` — or, later on, using the web page