Why study / use Javascript?

• pretty easy to start with
• easy to do useful things with it
• all browsers process Javascript
  - can use it in your own web pages
  - can understand what other web pages are doing (and steal from them if desired)
• ideas carry over into other languages

• there are good reasons not to use Javascript too:
  - limited functionality for general use, outside of web pages
  - many irregularities and surprising behaviors
  - no browsers match ostensible standards exactly
  - doesn’t illustrate much about how big programs are built

Javascript components

• Javascript programming language
  - statements that tell the computer what to do
    get user input, display output,
    set values, do arithmetic,
    test conditions, branch, loop, ...

• libraries, built-in functions
  - pre-fabricated pieces that you don’t have to create yourself
    math functions, text manipulation

• access to browser and web pages
  - buttons, text areas, images, page contents, ...
Basic example: add 2 numbers

- Javascript code between `<script>`...`</script>` tags

```html
<html>
<body>

<P> add2.html: adds 2 numbers
<script>
    var num1, num2, sum
    num1 = prompt("Enter first number")
    num2 = prompt("Enter second number")
    sum = parseInt(num1) + parseInt(num2)
    alert("Sum = " + sum)
</script>

</body>
</html>
```

Variation: concatenate two strings of characters

```html
<html>
<body>

<P> name2.html: concatenates 2 names
<script>
    var num1, num2, sum
    num1 = prompt("Enter last name")
    num2 = prompt("Enter first name ")
    sum = num2 + num1
    alert("hello, " + sum)
</script>

</body>
</html>
```
Adding up numbers: addup.html

- variables, operators, expressions, assignment statements
- while loop, relational operator

```html
<html>
<body>
<script>
  var sum = 0
  var num
  num = prompt("Enter new value, or 0 to end")
  while (num != 0) {
    sum = sum + parseInt(num)
    num = prompt("Enter new value, or 0 to end")
  }
  alert("Sum = " + sum)
</script>
</body>
</html>
```

Find the largest number: max.html

- needs an If to test whether new number is bigger
  - another relational operator
- needs parseInt or parseFloat to treat input as a number

```html
var max = 0
var num
num = prompt("Enter new value, or 0 to end")
while (num != 0) {
  if (parseFloat(num) > max)
    max = num
  num = prompt("Enter new value, or 0 to end")
}
document.write("<p> Max = " + max)
```
Programming language components

- **statements**: instructions that say what to do
- **variables**: places to hold data in memory while program is running
  - numbers, text, ...
- **syntax**: grammar rules for determining what's legal
  - what's grammatically legal? how are things built up from smaller things?
- **semantics**: what things mean
  - what do they compute?

- most languages are higher-level and more expressive than the assembly language for the toy machine
  - statements are much richer, more varied, more expressive
  - variables are much richer, more varied
  - grammar rules are more complicated
  - semantics are more complicated
- but it's basically the same idea

Variables, constants, expressions, operators

- a **variable** is a place in memory that holds a value
  - has a name that the programmer gave it, like sum or Area or n
  - in Javascript, can hold any of multiple types, most often numbers like 1 or 3.14, or sequences of characters like "Hello" or "Enter new value"
  - always has a value
  - has to be set to some value initially before it can be used
  - its value will generally change as the program runs
  - ultimately corresponds to a location in memory
  - but it's easier to think of it just as a name for information
- a **constant** is an unchanging literal value like 3 or "hello"
- an **expression** uses operators, variables and constants to compute a value
  - 3.14 * rad * rad
- **operators** include + - * /
Computing area: area.html

```javascript
var rad, area;
rad = prompt("Enter radius")
while (rad != null) {
    area = 3.14 * rad * rad
    document.write("<p> radius = " + rad + ", area = " + area)
    rad = prompt("Enter radius")
}
```

- how to terminate the loop
  - 0 is a valid data value
  - prompt returns null for Cancel and "" for OK without typing

- string concatenation to build up output line
- no exponentiation operator so we use multiplication

Types, declarations, conversions

- variables have to be declared in a `var` statement

- each variable holds information of a specific type
  - really means that bits are to be interpreted as info of that type
  - internally, 3 and 3.00 and "3.00" are represented differently

- Javascript usually infers types from context, does conversions automatically
  - "radius = " + rad

- sometimes we have to be explicit:
  - parseInt(string) if can’t tell from context that string is meant as an integer
  - parseFloat() if it could have a fractional part
Errors:

- Javascript is very bad at reporting errors!
- if you do something wrong, the browser may not tell you at all
- if you use Mozilla, turn on the Javascript console (Tools)

Control flow statements: decisions and loops

- if-else is the Javascript version of compare and goto

```javascript
if (condition is true) {
  do this part
} else {
  do this part instead
}
```

- while is a Javascript version of a loop

```javascript
while (condition is true) {
  do these statements
}
```
if-else examples (sign.html)

```javascript
if (i >= 0) {
    alert(i + " is positive")
}

if (i >= 0) {
    alert(i + " is positive")
} else {
    alert(i + " is negative")
}

• can include else-if sections for a series of decisions:
  if (i > 0) {
      print i, " is greater than zero"
  } else if (i == 0) { // note: ==
      alert(i + " is zero")
  } else {
      alert(i + " is negative")
  }
```

Control flow statements: while loop

• counting or "indexed" loop:
  ```javascript
  i = 1
  while (i <= 10) {
      do something with i
      i = i + 1
  }
  ```

• the most general loop; can simulate all others
  ```javascript
  var n = prompt("Enter number")
  while (n != null) {
      i = 0
      while (i <= n) {
          document.write("<br>" + i + " " + i*i)
          i = i + 1
      }
      n = prompt("Enter number")
  }
  ```
Functions

- A function is a group of statements that does some computation
  - The statements are collected into one place and given a name
  - Other parts of the program can "call" the subroutine
    - That is, use it as a part of whatever they are doing
  - Can give it values to use in its computation (arguments or parameters)
  - Computes a value that can be used in expressions
  - The value need not be used

- Javascript provides some useful functions

- You can write your own functions

Function examples

- Syntax
  ```javascript
  function name (list of "arguments") {
    the statements of the function
  }
  ```

- Function definition:
  ```javascript
  function area(r) {
    return 3.14 * r * r
  }
  ```

- Function uses:
  ```javascript
  rad = prompt("Enter radius")
  alert("radius = " + rad + ", area = " + area(rad))
  ```
  ```javascript
  alert("area of ring =" + area(1.75) - area(0.6))
  ```
var r1, r2;
r1 = prompt("Enter radius 1")
while (r1 !== null) {
    r2 = prompt("Enter radius 2")
    alert("area = " + (area(r1) - area(r2))) // parens needed!
    r1 = prompt("Enter radius 1")
}

function area(r) {
    return 3.14 * r * r
}

Why use functions?

• if a computation appears several times in one program
  - a function collects it into one place
• breaks a big job into smaller, manageable pieces
  - that are separate from each other
• defines an interface
  - implementation details can be changed as long as it still does the same job
• multiple people can work on the program
• a way to use code written by others long ago and far away
  - most of Javascript’s library of useful stuff is accessed through functions
Javascript library functions, etc.

- **Math**
  - `sqrt`, `max`, `min`, `random`, ...
- **String**
  - searching, substring, case conversion, convert to HTML,
- "Regular expression"
  - pattern matching
- **Date/Time**
  - current time, elapsed time, conversions

- **Array**
  - set of related items, accessible by index
  - use for things like sorting

A working sort example

```javascript
var name, i = 0, j, temp
var names = new Array()

// fill the array with names
name = prompt("Enter new name, or OK to end")
while (name != ") { 
  names[names.length] = name
  name = prompt("Enter new name, or OK to end")
}

// insertion sort
for (i = 0; i < names.length-1; i++) {
  for (j = i+1; j < names.length; j++) {
    if (names[i] > names[j]) {
      temp = names[i]
      names[i] = names[j]
      names[j] = temp
    }
  }
}

// print names
for (i = 0; i < names.length; i++) {
  document.write("<br> " + names[i])
}
```
Summary: elements of (most) programming languages

- constants: literal values like 1, 3.14, "Error!"
- variables: places to store data and results during computing
- declarations: specify name (and type) of variables, etc.
- expressions: operations on variables and constants to produce new values
- assignment: store a new value in a variable
- statements: assignment, input/output, loop, conditional, call
- conditionals: compare and branch; if-else
- loops: repeat statements while a condition is true
- functions: package a group of statements so they can be called/used from other places in a program
- libraries: functions already written for you

How Javascript works

- recall the compiler -> assembler -> machine instruction process for Fortran, C, etc.
- Javascript is analogous, but differs significantly in details
- when the browser sees Javascript in a web page,
  - checks for errors (may or may not report them usefully)
  - compiles your program into instructions in an "assembly language" for something like the toy machine
    but richer, more complicated, higher level
  - runs a simulator program (like the toy demo) that interprets these instructions
- the simulator is usually called
  "interpreter" (older term) or "virtual machine" (newer, as in Java)