Programming language components

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

- 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

What is Javascript?

- a comparatively simple language that can be compiled and run within a browser (not true of Java, the competitor at the time)

- designed & implemented in 1995 by Brendan Eich at Netscape
- provides dynamic effects (e.g., drag and drop), local computation, effective and efficient interaction with server
- widely used
  - supported by all browsers
  - Javascript code on almost all web pages
  - increasingly used on servers (outside of browsers)

Javascript components

- Javascript language
  - statements that tell the computer what to do
    get user input, display output, set values, do arithmetic,
    test conditions, repeat groups of statements, …
- libraries, built-in functions
  - pre-fabricated pieces that you don't have to create yourself
    `alert`, **`prompt`**, math functions, text manipulation, ...
- access to browser and web pages
  - buttons, text areas, images, page contents, ...

- you are not expected to remember syntax or other details
- you are not expected to write code in exams
  (though a bit in problem sets and labs)

- you are expected to understand the ideas
  - how programming and programs work
  - figure out what a tiny program does or why it's broken

Basic example 0: echo a name (name.html)

- Javascript code appears in HTML file between <script> tags
  `<script> ... </script>`
- this example shows a variable and a dialog box

```
<html>
<body>
<P> nam2.html: echoes a name
<script>
    var name;
    name = prompt("What's your name?");
    alert("hello, " + name);
</script>
```

Basic example 1: join 2 names (name2.html)

- Javascript code appears in HTML file between <script> tags
  `<script> ... </script>`
- shows variables, dialog boxes, and an operator

```
<html>
<body>
<P> name2.html: joins 2 names
<script>
    var firstname, secondname, result;
    firstname = prompt("Enter first name");
    secondname = prompt("Enter last name");
    result = firstname + secondname; // + means "join" here
    alert("hello, " + result);       // and here
</script>
```

Basic example 2: add 2 numbers (add2.html)

- dialog boxes, variables, arithmetic, conversion

```
<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); // "+" means "add"
  alert(sum);
</script>
```

`parseInt(...)` converts a sequence of characters into its integer value
there's also a parseFloat(…) for floating point numbers

Adding up lots of numbers: addup.html

- variables, operators, expressions, assignment statements
- while loop, relational operator (!= means "not equal to")

```
<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>
```

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 + - * /

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
  - "Sum = " + sum
- sometimes we have to be explicit:
  - `parseInt(...)` if can't tell from context that string is meant as an integer
  - `parseFloat(...)` if it could have a fractional part

Making decisions and repeating statements

- if-else statement makes decisions
  - the Javascript version of decisions written with ifzero, ifpos, ...

```
if (condition is true) {
      do this group of statements
} else {
    do this group of statements instead
}
```

- while statement repeats groups of statements
  - a Javascript version of loops written with ifzero and goto

```
while (condition is true) {
     do this group of statements
}
```

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 function
    that is, use it as a part of whatever they are doing
  - can give it values to use in its computation (arguments or parameters)
  - the function computes a value that can be used in expressions
  - the value need not be used
- Javascript provides some useful built-in functions
  - e.g., prompt, alert, ...
- you can write your own functions

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
- statements: assignment, conditional, loop, function call
  - assignment: store a new value in a variable
  - conditional: compare and branch; if-else
  - loop: 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