Eric Roberts and Jerry Cain
CS 106J

Handout #8
April 10, 2017

Simple JavaScript

Simple JavaScript

Eric Roberts
CS 106J
April 10, 2017

JavaScript and SJS

- JavaScript was developed at the Netscape Communications Corporation in 1995, reportedly by a single programmer in just 10 days. The language, which was called Mocha at the time, was designed to serve as a programming language that could be embedded in web pages viewed in the browser.
- JavaScript has since become the dominant language for all interactive web content and appears in some surveys as the most popular language in the computing industry.
- CS 106J uses a subset of JavaScript that we are calling ***SJS*** for ***Stanford JavaScript***. The SJS subset includes those parts of JavaScript that are important for learning the fundamentals of programming, which is the primary goal of all the CS 106 classes.

Arithmetic Expressions

- Like most languages, JavaScript specifies computation in the form of an ***arithmetic expression***, which consists of ***terms*** joined together by ***operators***.
- Each term in an arithmetic expression is one of the following:
  – An explicit numeric value, such as 2 or 3.14159265
  – A variable name that serves as a placeholder for a value
  – A function call that computes a value
  – An expression enclosed in parentheses
- The operators are typically the familiar ones from arithmetic:

`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`%`	Remainder

The Remainder Operator

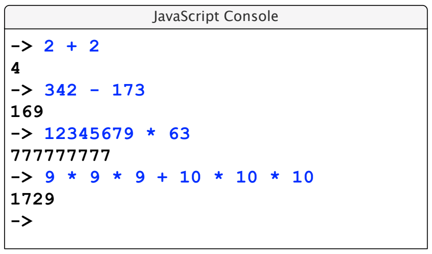
- The only arithmetic operator that has no direct counterpart in traditional mathematics is `%`, which computes the remainder when the first divided by the second:

`14 % 5`	*returns*	`4`
`14 % 7`	*returns*	`0`
`7 % 14`	*returns*	`7`

- The result of the `%` operator make intuitive sense only if both operands are positive. The examples in the book do not depend on knowing how `%` works with negative numbers.
- The remainder operator turns out to be useful in a surprising number of programming applications and is well worth a bit of study.

Using the SJS Console

- The easiest way to get a sense of how arithmetic expressions work is to enter them on the SJS console.


JavaScript Console

```
-> 2 + 2
4
-> 342 - 173
169
-> 12345679 * 63
777777777
-> 9 * 9 * 9 + 10 * 10 * 10
1729
->
```

Variables

- The simplest terms that appear in expressions are constant literals and variables. A ***variable*** is a placeholder for a value that can be updated as the program runs.
- A variable in JavaScript is most easily envisioned as a box capable of storing a value

`answer`

$\boxed{42}$

- Each variable has the following attributes:
  – A ***name***, which enables you to tell the variables apart.
  – A ***value***, which represents the current contents of the variable.
- The name of a variable is fixed; the value changes whenever you ***assign*** a new value to the variable.

Variable Declarations

- In JavaScript, you must ***declare*** a variable before you can use it. The declaration establishes the name of the variable and, in most cases, specifies the initial value as well.
- The most common form of a variable declaration is

```
var name = value;
```

  where *name* is an identifier that indicates the name of the variable, and *value* is an expression specifying the initial value.
- Most declarations appear as statements in the body of a function definition. Variables declared in this way are called ***local variables*** and are accessible only inside that function.

Constant Declarations

- It is often useful to give names to values that you don't intend to change while the program runs. Such values are called ***constants***.
- A constant declaration is similar to a variable declaration:

```
const name = value;
```

  As before, *name* is an identifier that indicates the name of the constant, and *value* is an expression specifying its value.

Naming Conventions

- In JavaScript, all names must conform to the syntactic rules for identifiers, which means that the first character must be a letter and the remaining characters must be letters, digits, or the underscore character.
- Beyond these rules that apply to all JavaScript names, there are several conventions that programmers use to make their identifier names easier to recognize:
  - Variable names and function names begin with a lowercase letter. If a name consists of more than one word, the first letter in each word is capitalized, as in `numberOfStudents`. This convention is called ***camel case***.
  - Class names and program names begin with an uppercase letter.
  - Constant names are written entirely in uppercase and use the underscore character to separate words, as in `MAX_HEADROOM`.

Precedence

- If an expression contains more than one operator, JavaScript uses ***precedence rules*** to determine the evaluation order. The arithmetic operators have the following relative precedence:

unary −	*highest*
`*` `/` `%`	↕
`+` `-`	*lowest*

  Thus, JavaScript evaluates unary `-` operators first, then the operators `*`, `/`, and `%`, and then the operators `+` and `-`.
- Precedence applies only when two operands compete for the same operator. If the operators are independent, JavaScript evaluates expressions from left to right. Parentheses may be used to change the order of operations.

Exercise: Precedence Evaluation

What is the value of the expression at the bottom of the screen?

```
1 * 2 * 3 + (4 + 5) % 6 * (7 + 8) - 9
```

Assignment Statements

- You can change the value of a variable in your program by using an ***assignment statement***, which has the general form:

```
variable = expression;
```

- The effect of an assignment statement is to compute the value of the expression on the right side of the equal sign and assign that value to the variable that appears on the left. Thus, the assignment statement

```
total = total + value;
```

  adds together the current values of the variables `total` and `value` and then stores that sum back in the variable `total`.
- When you assign a new value to a variable, the old value of that variable is lost.

Shorthand Assignments

- Statements such as

```
total = total + value;
```

are so common that JavaScript allows the following shorthand:

```
total += value;
```

- The general form of a ***shorthand assignment*** is

```
variable op= expression;
```

where *op* is any of JavaScript's binary operators. The effect of this statement is the same as

```
variable = variable op (expression);
```

Increment and Decrement Operators

- Another important shorthand that appears frequently in JavaScript programs is the ***increment operator***, which is most commonly written immediately after a variable, like this:

```
x++;
```

The effect of this statement is to add one to the value of `x`, which means that this statement is equivalent to

```
x += 1;
```

or

```
x = x + 1;
```

- The `--` operator (which is called the ***decrement operator***) is similar but subtracts one instead of adding one.

Functions Revisited

- Last week, you learned that a ***function*** in Karel is a sequence of statements that has been collected together and given a name.
- Although that definition also applies in JavaScript, it fails to capture the idea that functions can process information.
- In JavaScript, a function can take information from its caller, perform some computation, and then return a result.
- This notion that functions exist to manipulate information and return results makes functions in programming similar to functions in mathematics, which is historically the reason for the name.

Functions in Mathematics

- The graph at the right shows the values of the function

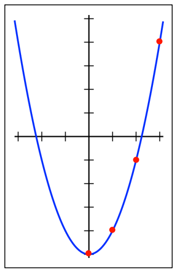
$$f(x) = x^2 - 5$$

- Plugging in a value for x allows you to compute the value of $f(x)$, as follows:

$$f(0) = 0^2 - 5 = -5$$
$$f(1) = 1^2 - 5 = -4$$
$$f(2) = 2^2 - 5 = -1$$
$$f(3) = 3^2 - 5 = 4$$

- The JavaScript version of $f(x)$ is

```
function f(x) {
   return x * x - 5;
}
```



Writing JavaScript Functions

- The general form of a function definition is

```
function name(parameter list) {
   statements in the function body
}
```

where *name* is the name of the function, and *parameter list* is a list of variables used to hold the values of each argument.

- You can return a value from a function by including a `return` statement, which is usually written as

```
return expression;
```

where *expression* is an expression that specifies the value you want to return.

Examples of Simple Functions

- The following function converts Fahrenheit temperatures to their Celsius equivalent:

```
function fahrenheitToCelsius(f) {
   return 5 / 9 * (f - 32);
}
```

- The following function computes the area of a triangle from its base and height:

```
function triangleArea(base, height) {
   return (base * height) / 2;
}
```

Exercise: Money in the Wizarding World

In J. K. Rowling's *Harry Potter* books, wizards use a currency with the following equivalences:

29 knuts	=	1 sickle
17 sickles	=	1 galleon

Write a function

```
function numberOfGalleons(knuts, sickles)
```

that returns the number of galleons (which will usually have a decimal fraction) corresponding to the specified number of knuts and sickles. For example, calling

```
numberOfGalleons(29 * 17, 17)
```

should return 2.

Useful Functions in the `Math` Class

`Math.PI`	The mathematical constant π
`Math.E`	The mathematical constant e
`Math.abs(`x`)`	The absolute value of x
`Math.max(`x`, `y`, . . .)`	The largest of the arguments
`Math.min(`x`, `y`, . . .)`	The smallest of the arguments
`Math.round(`x`)`	The closest integer to x
`Math.floor(`x`)`	The largest integer not exceeding x
`Math.log(`x`)`	The natural logarithm of x
`Math.exp(`x`)`	The inverse logarithm (e^x)
`Math.pow(`x`, `y`)`	The value x raised to the y power (x^y)
`Math.sin(`θ`)`	The sine of θ, measured in radians
`Math.cos(`θ`)`	The cosine of θ, measured in radians
`Math.random(`x`)`	A random number between 0 and 1

Exercise: Calculating a Quotient

It is often useful to be able to compute the ***quotient*** of two numbers, which is the whole number that results if you divide the first by the second and then throw away any remainder. The usual way to calculate a quotient is to use `Math.floor` like this:

```
function quotient(x, y) {
   return Math.floor(x / y);
}
```

Rewrite the `quotient` function so that it uses only the standard arithmetic operators. Your function may assume that both x and y are positive.