

# From Pseudocode to “Real” Code

- Once we have expressed an algorithm in pseudocode, we need one more step to turn it into something that machines can do for us: conversion into an actual programming language, or “real” code
- For this course, that programming language is JavaScript — chosen because it is built-in to most Web browsers, which means you already have it on whatever computer you may be using
- This handout hopes to serve as a guide for converting pseudocode into JavaScript

## Pseudocode vs. Programming Languages

Unlike pseudocode, programming language code is meant to be “understood” and run by the computer — this is where the rubber meets the road:

- Programming language code is *much* more precise (and thus less flexible and less “forgiving”) than pseudocode
- Programming languages may have their own distinct symbols and “look,” which might vary significantly from the original pseudocode
- Programming languages may have multiple variations for the same concept (e.g., repetitions, conditionals)

# Naming and Comments in JavaScript

- The table below shows how our previous pseudocode notation translates into JavaScript
- They are similar, with JavaScript needing some additional symbols at times, such as semi-colons and braces

Pseudocode	JavaScript
name ← value	var name = value;
procedure name(input1, input2, ...) <u>algorithm body</u>	var name = function(input1, input2, ...) { <u>algorithm body</u> };
// Comment.	// One-line comment, or... /* Comment consisting of multiple lines. */

In all cases, include var only for the **first** time that you assign an expression to a name.

# Repetitions and Conditionals

Pseudocode	JavaScript
while (condition) <u>(code to repeat)</u>	while (condition) { <u>code to repeat</u> }
list ← [first, second, ...] for each (member in list) <u>(code to repeat)</u>	var list = [first, second, ...]; for (var index = 0; index < list.length; index += 1) { var member = list[index]; <u>code to repeat</u> }
if (condition) then <u>(code if condition is true)</u>	if (condition) { <u>code if condition is true</u> }
if (condition) then <u>(code if condition is true)</u> else <u>(code if condition is false)</u>	if (condition) { <u>code if condition is true</u> } else { <u>code if condition is false</u> }

# [Some] Built-In Operations

Pseudocode	JavaScript
← (assign an expression to a name)	=
+ (addition), - (subtraction)	+, -
× (multiplication), ÷ (division)	*, /
= (equal to), <> (not equal to)	===, !==
<, <= (less than [or equal to])	<, <=
>, >= (greater than [or equal to])	>, >=
integer division (no remainder)	parseInt(dividend / divisor)
remainder after division (modulo)	% (e.g., “((x % 2) === 1)” tests whether x is odd)
random number from min-max	Math.round((max - min) * Math.random()) + min

# Returning Answers and Invoking Other Algorithms

Pseudocode	JavaScript
return result	return result;
<pre> procedure algorithm(input)   <u>code for algorithm</u> ... algorithm(actualInput) ...           </pre>	<pre> var algorithm = function(input) {   <u>code for algorithm</u> }; ... algorithm(actualInput); ...           </pre>
<pre> procedure partialAnswer(input)   <u>code for partialAnswer</u>   return output value ← partialAnswer(someInput)           </pre>	<pre> var partialAnswer = function(input) {   <u>code for partialAnswer</u>   return output; }; var value = partialAnswer(someInput);           </pre>

In all cases, include var only for the **first** time that you assign an expression to a name.

# Lists (a.k.a. Arrays)

Pseudocode	JavaScript
// Creating an empty list. emptyList ← [ ]	/* 2 choices: */ var emptyList = [ ]; /* or: */ var emptyList = new Array();
// Accessing or assigning an item. item ← list[index] list[index] ← value	var item = list[index]; list[index] = value;
size ← number of items in a list	var size = list.length;
add value to list	list.push(value);
sort list “lexically,” ascending	list.sort(); // Caution: “a” comes after “Z”!
sort list numerically, ascending	list.sort(function(a, b) { return a - b; });
number ← smallest number in list	var number = Math.min.apply(Math, list);

In all cases, include var only for the **first** time that you assign an expression to a name.

# Interacting with the User

Pseudocode	JavaScript
input ← information from user (prompted by a message)	var input = prompt(message);
display message	alert(message);

The examples below work only for the course’s JavaScript scratch page:

retrieve text entered into the “Input 1” field on the JavaScript scratch page	var form = document.getElementById("scratch"); var input1Field = form.input1; var input1Text = input1Field.value;
display message at the bottom of the JavaScript scratch page	var displayBox = document.getElementById("display"); displayBox.innerHTML = message;

# Example Conversions from Pseudocode to JavaScript

- There's much more to JavaScript (especially with regard to what's "built-in") than shown here, but the preceding tables should be enough to translate the pseudocode that you've seen so far into real programs that you can run within a browser
- The overall approach would be:
  - ◊ Write out your pseudocode, and test it by hand to make sure that it does produce the expected results
  - ◊ Refer to the preceding tables to convert each pseudocode segment into its JavaScript equivalent

Pseudocode	JavaScript
<pre> procedure countCoins(amount, denomination)   currentAmount ← amount   coinCount ← 0   while (currentAmount ≥ denomination)     (coinCount ← coinCount + 1      currentAmount ← currentAmount - denomination)   return coinCount  procedure makeChange(amount)   currentAmount ← amount   quarters ← countCoins(currentAmount, 25)   currentAmount ← currentAmount - (25 × quarters)   dimes ← countCoins(currentAmount, 10)   currentAmount ← currentAmount - (10 × dimes)   nickels ← countCoins(currentAmount, 5)   currentAmount ← currentAmount - (5 × nickels)   pennies ← countCoins(currentAmount, 1)   return [quarters, dimes, nickels, pennies] </pre>	<pre> var countCoins = function(amount, denomination) {   var currentAmount = amount;   var coinCount = 0;   while (currentAmount &gt;= denomination) {     coinCount = coinCount + 1;     currentAmount = currentAmount - denomination;   }   return coinCount; };  var makeChange = function(amount) {   var currentAmount = amount;   var quarters = countCoins(currentAmount, 25);   currentAmount = currentAmount - (25 * quarters);   var dimes = countCoins(currentAmount, 10);   currentAmount = currentAmount - (10 * dimes);   var nickels = countCoins(currentAmount, 5);   currentAmount = currentAmount - (5 * nickels);   var pennies = countCoins(currentAmount, 1);   return [quarters, dimes, nickels, pennies]; }; </pre>
<pre> procedure listRPM(factor1, factor2)   if (factor1 &gt; factor2) then     (term1 ← factor2      term2 ← factor1)   else     (term1 ← factor1      term2 ← factor2)    addendList ← [ ]   while (term1 &gt; 0)     (if (term1 is odd) then      (add term2 to addendList)      term1 ← halveWithoutRemainder(term1)      term2 ← double(term2))    product ← 0   for each (number in addendList)     (product ← product + number)   return product </pre>	<pre> var listRPM = function(factor1, factor2) {   var term1 = factor1; var term2 = factor2;   if (factor1 &gt; factor2) {     term1 = factor2; term2 = factor1;   }   var addendList = [ ];   while (term1 &gt; 0) {     if ((term1 % 2) == 1) {       addendList.push(term2);     }     term1 = parseInt(term1 / 2);     term2 = term2 * 2;   }   var product = 0;   for (var index = 0; index &lt; addendList.length; index += 1) {     product = product + addendList[index];   }   return product; }; </pre>