EECS 368
Programming Language Paradigms

David O. Johnson

Fall 2022

Reminders

- Assignment 1 due: 11:59 PM, Wednesday, September 7
- Assignment 2 due: 11:59 PM, Monday, September 19

Any Questions?

In-Class Problem Solution

- 2-(8-26) In-Class Problem Solution.pptx

Any Questions?

JavaScript Syntax

- JavaScript syntax is very similar to C++ syntax.
- C++ syntax evolved from C syntax.
- Java also uses a syntax similar to C and C++.
- JavaScript, Java, C++, C, and C# are sometimes called “curly brace” languages because they all have a similar syntax based on the original C programming language.
- If in doubt, use C++ syntax …
- But beware there are a few subtle differences!

JavaScript Syntax

- Statements in JavaScript are delimited by semicolons (;):

```
console.log(ten * ten);
```

- Curly braces { } are used like they are in C++ as we will see in a moment.

```
if (num < 10) {
  console.log("Small");
} else if (num < 100) {
  console.log("Medium");
} else {
  console.log("Large");
}
```

- Curly braces {} serve the same purpose as indenting in Python.
- Like C++, indenting is not required in JavaScript, but highly recommended to show code inside curly braces.

JavaScript Comments

- Single line comments are preceded by //

```
//This is a comment.
```

- Multi-line comments are delimited by /* and */

```
/*
  I first found this number scrawled on the back of an old
  notebook. Since then, it has often dropped by, showing up in
  phone numbers and the serial numbers of products that I've
  bought. It obviously likes me, so I've decided to keep it.
*/
```

Any Questions?

JavaScript Programs

- JavaScript programs are composed of:
  - Functions
  - Variables (or bindings)
  - Control Flow
- We will talk about each of these individually.

Functions

- A function in JavaScript is a set of statements that performs a task or calculates a value.
- It should take some input and return an output where there is some obvious relationship between the input and the output.
- Note that we have three functions:
  - preload
  - create
  - update
- () – contains any variables or arguments passed to the function as input
- {} – contains code for functions
- There is no code yet

```
var config = {
    type: Phaser.AUTO,
    width: 800,
    height: 600,
    scene: {
        preload: preload,
        create: create,
        update: update
    }
};

var game = new Phaser.Game(config);

function preload ()
{
}

function create ()
{
}

function update ()
{
}
```

Return Values

- Functions may also produce values.
- For example, the function Math.max takes any amount of number arguments and gives back the greatest:

```
console.log(Math.max(2, 4));
// → 4
```

- When a function produces a value, it is said to return that value.
- Function calls can be used within larger expressions:

```
console.log(Math.min(2, 4) + 100);
// → 102
```

- Later we will learn how to write your own functions.

Any Questions?

JavaScript Variables

- Store information used by the functions.
- This program has 2 variables.
- The variables are defined outside of the three functions, i.e., not within the {}
- These are referred to as a global variables.
- All 3 functions can see them.
- A variable defined inside a function i.e., within the {} is called a local variable.
- Local variables can only be seen by the function in which it is defined.

```
var config = {
    type: Phaser.AUTO,
    width: 800,
    height: 600,
    scene: {
        preload: preload,
        create: create,
        update: update
    }
};

var game = new Phaser.Game(config);

function preload ()
{
}

function create ()
{
}

function update ()
{
}
```

Bindings (or Variables)

- To catch and hold values, JavaScript provides a thing called a binding, or variable:
- JavaScript has two ways to create a binding that is changeable, let and var:

```
let ten = 10;
let one = 1, two = 2;
var name = "Ayda";
```

- const defines a permanent binding, which points at the same value for as long as it lives.
- This is useful for bindings that give a name to a value so that you can easily refer to it later.

```
const greeting = "Hello ";
```

- You can think of var and let as giving a temporary binding to a variable.
- And const as giving a permanent binding to a variable.

Any Questions?

Naming Conventions

- Function and variable names can include any alphabetic or numeric character, but the name must not start with a digit.
- Names may include dollar signs ($) or underscores (_) but no other punctuation or special characters.
- Names may not contain spaces, yet it is often helpful to use multiple words to clearly describe what the function or variable represents:

  fuzzylittleturtle

  fuzzy_little_turtle

  FuzzyLittleTurtle

  fuzzyLittleTurtle

Naming Conventions

- Words with a special meaning, such as let, are keywords, and they may not be used as names.
- There are also a number of words that are “reserved for use” in future versions of JavaScript, which also can’t be used as names.
- The full list of keywords and reserved words is rather long:

break case catch class const continue debugger default delete do else enum export extends false finally for function if implements import interface in instanceof let new package private protected public return static super switch this throw true try typeof var void while with yield

- Don’t worry about memorizing this list.
- If creating a function and variable produces an unexpected syntax error, see whether you’re trying to define a reserved word.

Any Questions?

JavaScript Program Control Flow

- JavaScript contains the same control flow statements as C++ and most other languages:

  if then

  if then else

  switch

  for loop

  while

  do while

JavaScript if-then Statement

- The if-then statement is the most basic of all the control flow statements.
- It tells your program to execute a certain section of code *only if* a particular test evaluates to true.
- If this test evaluates to false, execution jumps to the end of the if-then statement.

```
if (this part is true) {
   // then execute the code between the {}
   // otherwise skip it
}
```

- All computer programming languages have some form of an if-then control structure.

JavaScript if-then-else Statement

- The if-then-else statement provides a secondary path of execution when an "if" clause evaluates to false.

```
if (this part is true) {
   // then execute the code between these {}
} else {
   // execute the code between these {}
   // if the part between the () is false
}
```

JavaScript else-if Statement

- The else clause can contain another if-then-else clause

```
if (this part is true) {
    // then execute the code between these {}
} else {
    if (this part is true) {
        // then execute the code between these {}
    } else {
        // execute the code between these {}
        // if the part between the () is false
    }
}
```

- This can be shortened to:

```
if (this part is true) {
    // then execute the code between these {}
} else if (this part is true) {
    // execute the code between these {}
    // the part between the () is false
} else {
    // execute the code between these {}
    // if the part between the () is false
}
```

JavaScript switch Statement

- It is not uncommon for code to look like this:

```
if (x == "value1") action1();
else if (x == "value2") action2();
else if (x == "value3") action3();
else defaultAction();
```

- As in C++, a construct called switch is intended to express such a control flow in a more direct way.

```
switch (prompt("What is the weather like?")) {
  case "rainy":
    console.log("Remember to bring an umbrella.");
    break;
  case "cloudy":
    console.log("Go outside.");
    break;
  default:
    console.log("Unknown weather type!");
    break;
}
```

JavaScript for-loop Statement

- The for-loop is often the tool you will use when you want to create a loop.
- The for-loop has the following syntax:

- for (*statement 1*; *statement 2*; *statement 3*) {
  code block to be executed
  }

- **Statement 1** is executed before the loop (the code block) starts.
- **Statement 2** defines the condition for running the loop (the code block).
- **Statement 3** is executed each time after the loop (the code block) has been executed.
- Example
- for (var i = 0; i < 5; i++) {
  text += "The number is " + i + "
";
  }

JavaScript while Statement

- The while-loop is another tool you will use when you want to create a loop.
- The while-loop has the following syntax:

- while (*logic_expression*) {
  code block to be executed
  }

- The word while is followed by a logic expression in parentheses and then a statement, much like if.
- The loop keeps entering the code block as long as the logic expression produces a value that gives true when converted to Boolean.

```
let number = 0;
while (number <= 12) {
  console.log(number);
  number = number + 2;
}
// → 0
// → 2
//   … etc.
```

JavaScript do-while Statement

- The do...while statements combo defines a code block to be executed once, and repeated as long as a condition is true.
- The do...while is used when you want to run a code block at least one time.

```
do {
  code block to be executed
}
while (logic_expression);
```

- For example:

```
let text = "";
let i = 0;
do {
  text += i + "<br>";
  i++;
}
while (i < 5);
// → 4
```

Breaking Out of a Loop

- Having the looping condition produce false is not the only way a loop can finish.
- There is a special statement called break that has the effect of immediately jumping out of the enclosing loop.

```
for (let current = 20; ; current = current + 1) {
  if (current % 7 == 0) {
    console.log(current);
    break;
  }
}
// → 21
```

- The continue keyword is similar to break, in that it influences the progress of a loop.
- When continue is encountered in a loop body, control jumps out of the body and continues with the loop's next iteration.

Assignment Shortcuts

- Especially when looping, a program often needs to "update" a variable to hold a value based on that variable's previous value.

```
counter = counter + 1;
```

- JavaScript provides a shortcut for this.

```
counter += 1;
```

- This allows us to shorten our counting example a little more.

```
for (let number = 0; number <= 12; number += 2) {
  console.log(number);
}
```

- Similar shortcuts work for many other operators, such as result *= 2 to double result or counter -= 1 to count downward or line += "-" to add a character to a string.
- For counter += 1 and counter -= 1, there are even shorter equivalents: counter++ and counter--.

Summary

- You now know that a program is built out of statements, which themselves sometimes contain more statements.
- Statements tend to contain expressions, which themselves can be built out of smaller expressions.
- Putting statements after one another gives you a program that is executed from top to bottom.
- You can introduce disturbances in the flow of control by using conditional (if, else, and switch) and looping (while, do, and for) statements.
- Variables can be used to file pieces of data under a name, and they are useful for tracking state in your program.
- Functions are special values that encapsulate a piece of program.
- You can invoke them by writing functionName(argument1, argument2).
- Such a function call is an expression and may produce a value.

Any Questions?

In-Class Problem

- Show the output for this 2 line program:

```
for (let line = "-"; line.length < 7; line += "-")
  console.log(line);
```

- Hint: “line.length” gives the length of the string named “line”.
- Add comments to describe what each part of the for statement is doing.

If you have access to a web browser console, try to figure out the answer before checking it with the console.

- You may work together.
- You may ask me or one of the SIs for help.
- Submit your solution as a PDF to Canvas before 11:59 PM today.
- You may leave when you are done.