

# EECS1012

## Net-centric Introduction to Computing

### Lecture 8

## Introduction to JavaScript

---

#### **Acknowledgements**

Contents are adapted from web lectures for “Web Programming Step by Step”, by M. Stepp, J. Miller, and V. Kirst.

Slides have been ported to PPT by Dr. Xenia Mountroudou.

These slides have been edited for EECS1012, York University.

The contents of these slides may be modified and redistributed, please give appropriate credit.

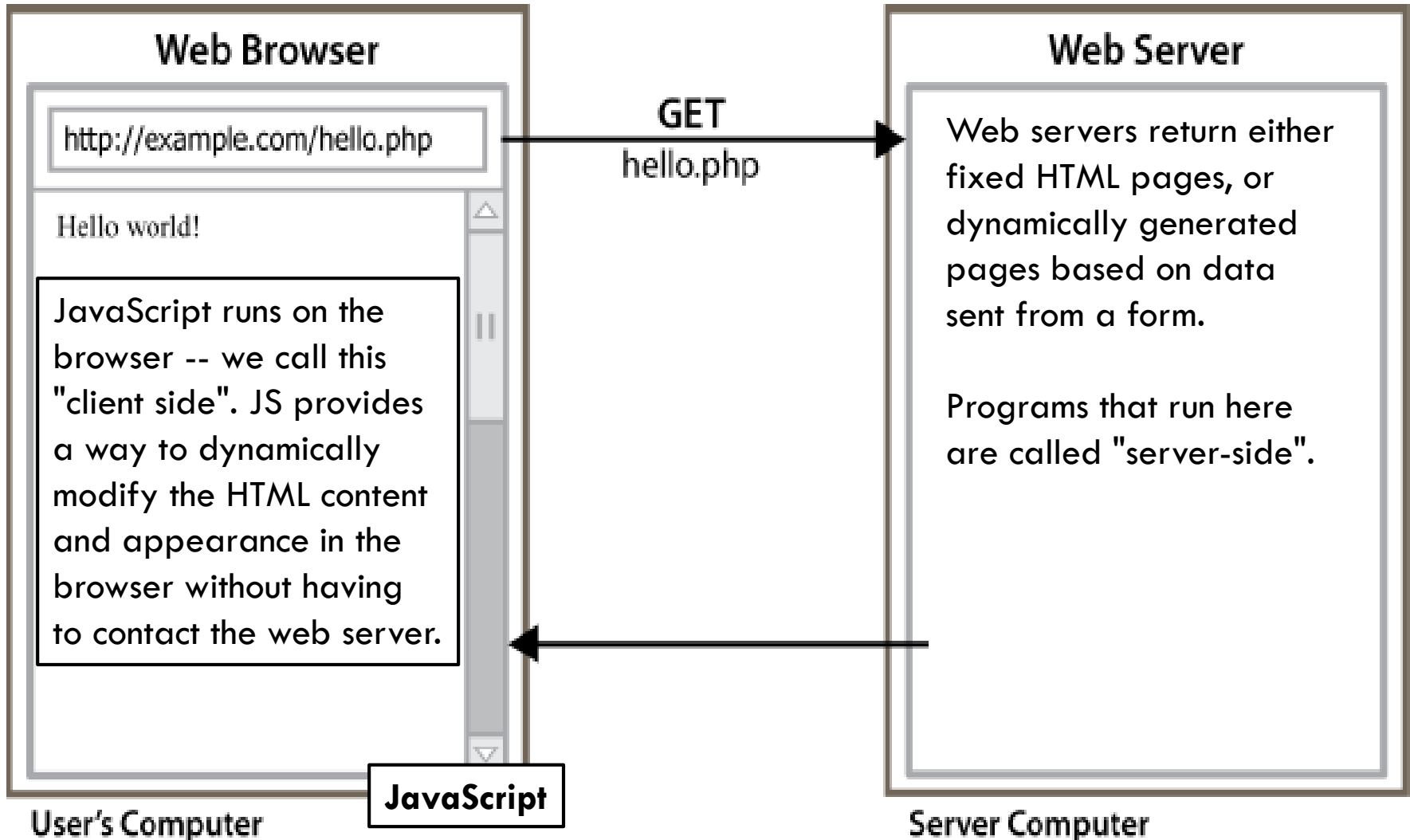
(Creative Commons) Michael S. Brown, 2017.

2

# JavaScript relationship to HTML

# Client-side scripting

3



# Why use client-side programming?

4

## Why use client-side scripting?

- client-side scripting benefits:
  - ▣ **usability:** can modify a page without having to post back to the server (faster UI)
  - ▣ **efficiency:** can make small, quick (dynamic) changes to page without waiting for server
  - ▣ **event-driven:** can respond to user actions like clicks and key presses

# What is JavaScript?

5

- a lightweight programming language ("scripting language")
  - ▣ used to make web pages interactive
  - ▣ insert dynamic text into HTML (ex: user name)
  - ▣ **react to events** (ex: page load or user click)
  - ▣ can get information about a user's computer (ex: browser type, history, etc)
  - ▣ perform calculations on user's computer (e.g.: for form validation)

# JavaScript (JS) vs. Java

6

- JavaScript is interpreted, Java is compiled
- JavaScript has more relaxed syntax and rules
  - ▣ fewer and "looser" data types
  - ▣ errors often silent (few exceptions)
- JS is contained within a web page and integrates with its HTML/CSS content

# JavaScript vs. Java

7



(JavaScript is a "mellow" version of Java)

Interestingly, even though the name has "Java" in it, JavaScript is not affiliated with Java (from Sun Microsystems – that is now part of the Oracle Corporation).

# Linking to a JavaScript file: `script`

8

```
<head>  
<script src="filename" type="text/javascript"></script>  
...  
</head>
```

*HTML*

- ❑ script tag should be placed in HTML page's **head**
- ❑ script code is stored in a separate .js file
- ❑ It is preferred to link in a JS file.
- ❑ Pay attention to the notation, there is an open and closed script tag `<script></script>`.



# HTML + CSS + JavaScript

9

- Just like a CSS file, we "link" to our JS file.

## HTML File

```
<head>  
<link href="my.css">  
<script src="my.js">  
</script>  
</head>
```

Defines the elements and overall structure of the webpage.

CSS file: my.css

**Defines the style of the webpage.**

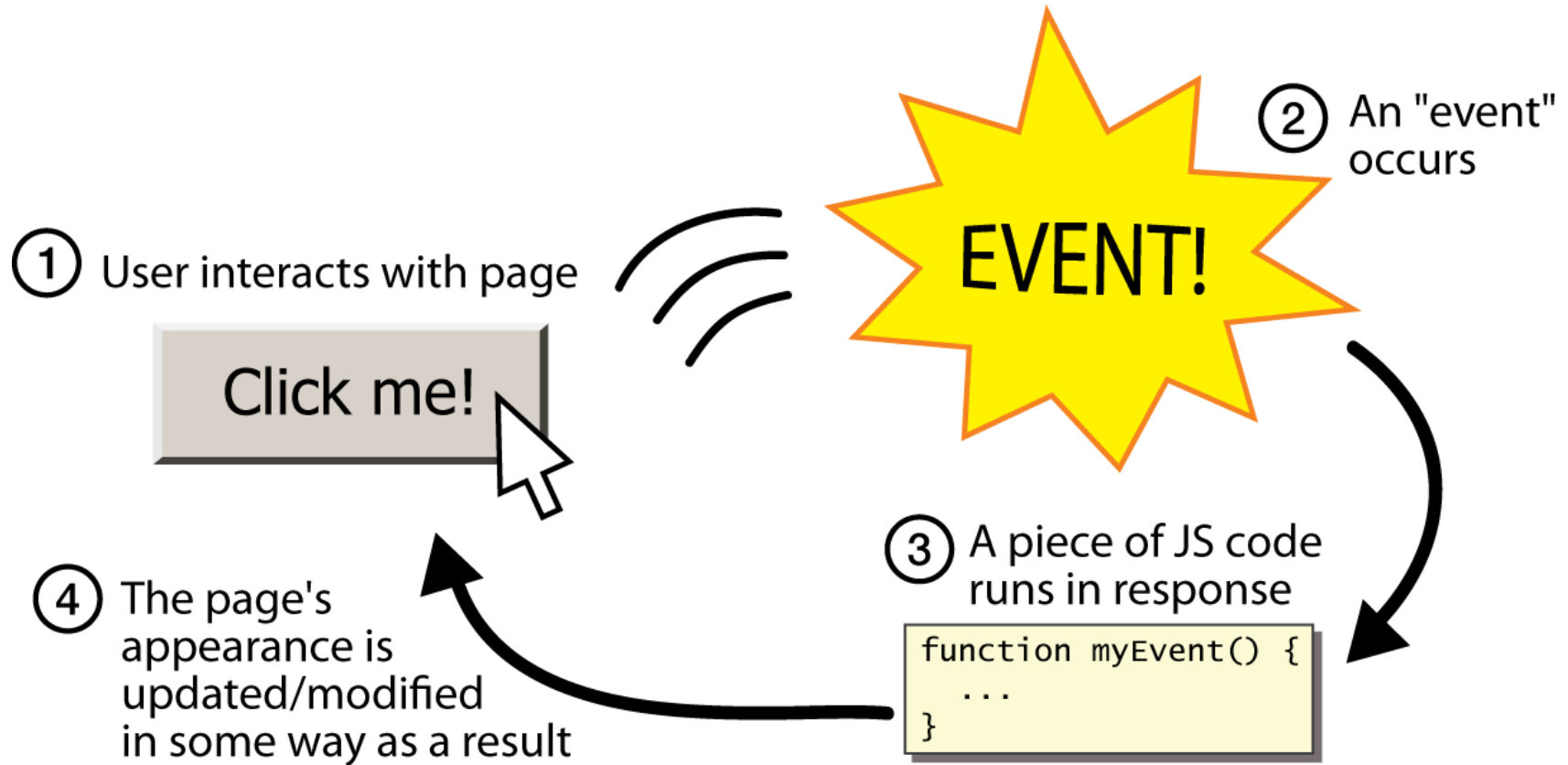
Javascript file: my.js

**Defines the interaction or behavior of the webpage.**

We can link in one or more JS files that provides functions that we can use to help make our webpage more dynamic.

# Event-driven programming

10



# Event-driven programming

11

- Event-driven programming: writing programs driven by user events
- Many programs (e.g. Java, C++, PHP, Python) start when the program is started.
- JavaScript programs instead wait for browser or user actions called *events* and respond to them.
  - ▣ **Examples of events:**
    - When a page loads or closes (this can be thought of events caused by the browser)
    - When a button is clicked (this is an event caused by the user)

# An example: start with a button

12

```
<button>Click me!</button>
```

*HTML*

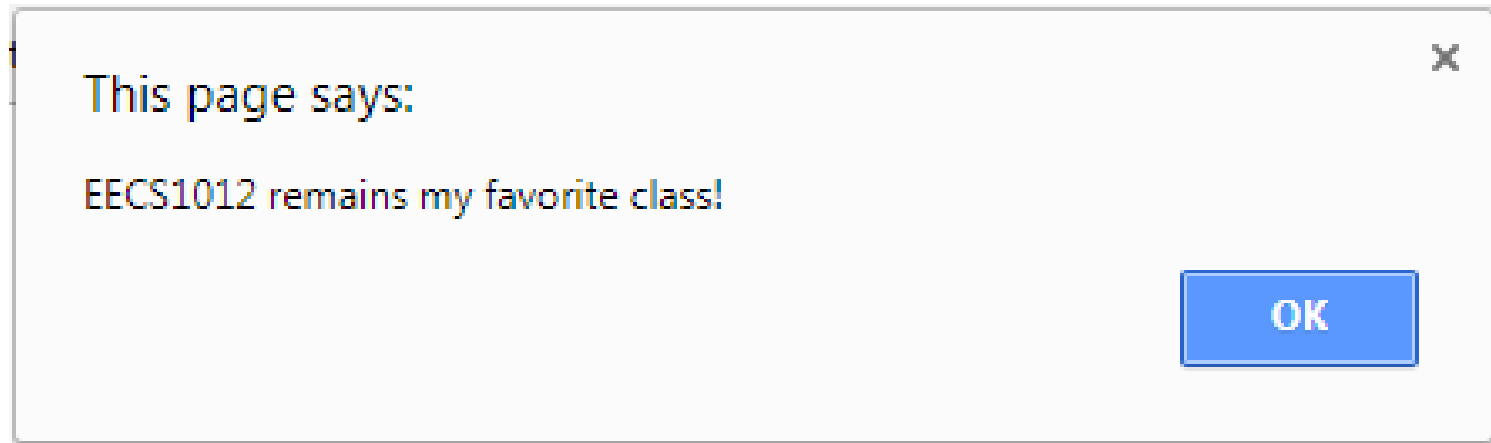
- button's text appears inside tag; can also contain images (note this is different than an HTML form)
- To make a responsive button or other UI control:
  1. choose the control (e.g. button) and event (e.g. mouse click) of interest
  2. write a JavaScript function to run when the event occurs
  3. attach the function to the event on the control

# Your first JavaScript statement: `alert`

13

```
alert("EECS1012 is my favorite class!");
```

*JS*



- a JS command that pops up a dialog box with a message
- The appearance of the "alert" may look different depending on the browser and operating system

# Defining a JavaScript function

14

```
/* the syntax to declare a function */  
function functionName() {  
    statement;  
    statement;  
    ...  
    statement;  
}
```

*JS*

```
/* an example of a function named "myFunction" */  
function myFunction() {  
    alert("EECS1012 remains my favorite class!");  
}
```

*JS*

- ❑ the above could be the contents of `example.js` linked to our HTML page
- ❑ statements placed into functions can be evaluated in response to user events.

# Event handlers in HTML

15

```
<element ... onclick="function();">...</element>
```

*HTML*

```
<button onclick="myFunction();">Click me!</button>
```

*HTML*

## Linking HTML to JavaScript

- JavaScript functions can be set as event handlers
  - ▣ when you interact with the element, the function will execute
- `onclick` is an HTML element property that can be set to call a JavaScript function
- We will see more examples, this example is just to get us started.

# Putting it together

16

## HTML FILE

```
<!DOCTYPE html>
<html>
<head>
  <script src="example.js"
type="text/javascript"></script>
</head>
<body>
  <button onclick="myFunction();" > Click Me!
</button>
</body>
</html>
```

Event (click) of the HTML button,  
calls the specified handler function

JS file: [example.js](#)

```
function myFunction() {
  alert("EECS1012 remains my favorite class!");
}
```

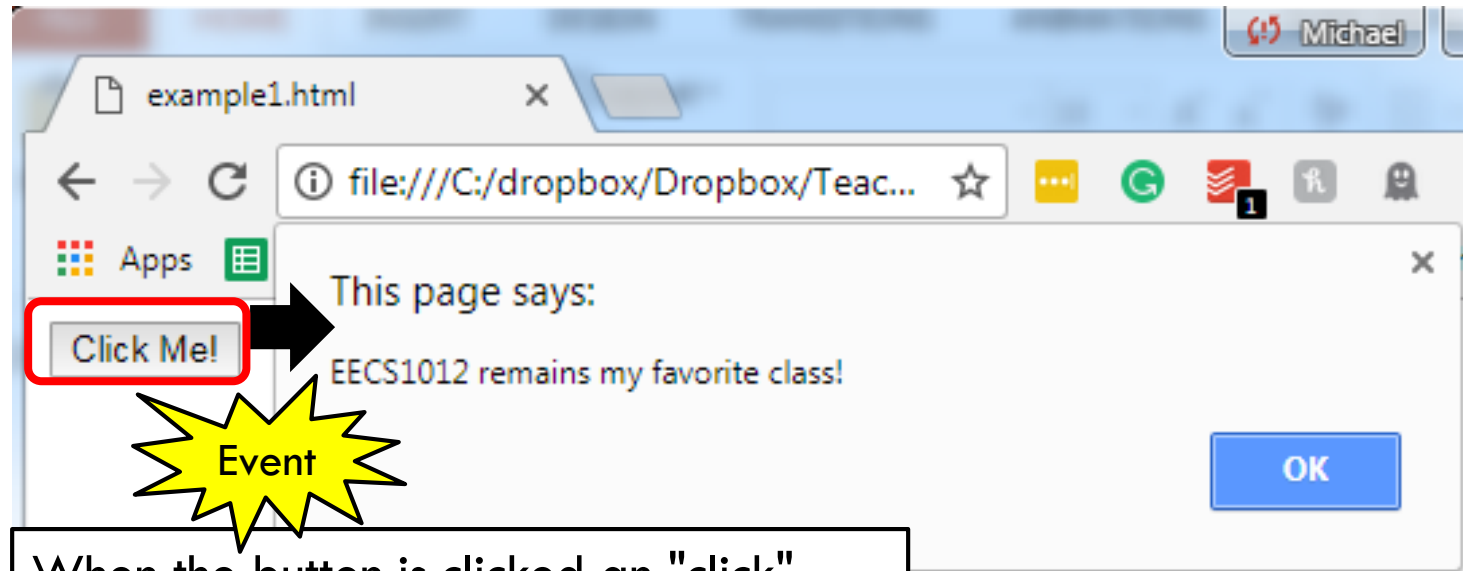
This statement links in the JS file. The JS file has our JavaScript code. Like CSS, this linking is done by file name.

Like CSS, the JS file has to be in the correct location to link it in.



# Effect of the previous code

17



When the button is clicked an "click" event occurs. This event calls the `myFunction()`. In the function, the `alert()` is called, causing this alert box to appear.

# Just an example

- The alert box example is a simple example to help us get started.
- It gives you an idea of how JavaScript programming is going to work.
- We will be writing functions that respond to events that happen on the page.
- Now, we need to understand the JavaScript language to write interesting functions.

19

# JavaScript Language

# JavaScript

20

- If you are familiar with other programming languages, you will likely be able to quickly pick up JavaScript
- In fact, most programming languages are quite similar and generally you can learn them quickly once you have experience in another language.

# Comments in JavaScript

21

- JavaScript comments are similar to CSS
- Comments are placed in `/* ..... */`
- You can also use `//` for single line comments.
- All comments are ignored by JavaScript.
- We will use comments in our notes

# JavaScript variables

22

```
var clientName = "Connie Client"; /* variable clientName */  
var age = 32; // variable age  
var id = 3994330; // variable id
```

- Variables are used to store and retrieved data.
- Variables are defined by the keyword **var**
- Variables are categorized into different types

# Rules for variable names

23

- First character must be a letter or an underscore (`_`)
- The rest of the variable can be any letter, number, or underscore
- Variable names are case sensitive
  - ▣ `age`, `Age`, `AGE` would all be **different** variable names
- You cannot use JavaScript reserved words for variable names
  - ▣ Example of a reserved word? `var`  
`var var = 10; // Since var is used to declare a variable`  
`// you can't use var as a variable`

# Variable name examples

24

## □ Valid names:

<code>_myVar</code>	<code>thissisalongvariablename</code>	<code>num</code>
<code>_var</code>	<code>eeecs1012</code>	<code>myString</code>
<code>name1</code>	<code>test_1</code>	<code>x</code>

## □ Invalid names:

<code>1test</code>	<code>/* starts with a number */</code>
<code>test 1</code>	<code>/* there is a space in the name */</code>
<code>t\$est</code>	<code>/* non alphanumeric character */</code>
<code>var</code>	<code>/* reserved word */</code>



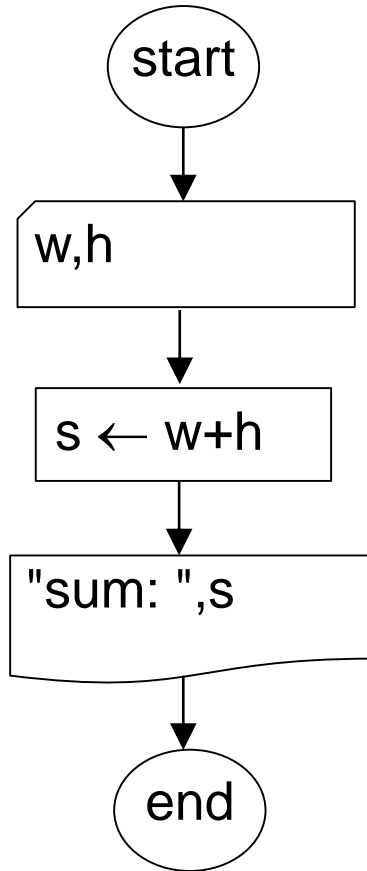
# JS data types

25

TYPE	Explanation	Example
Number	Integers and numbers with decimal places.	99, 2.8, 5, -10
String	A variable that is a collection of characters. Sometimes we call this a string literal.	“Hello”, “EECS1012”
Boolean	A variable that wholes only two possible values – <b>true</b> or <b>false</b> .	true or false
Array	A variable that is actually a collection of variables that can be access with an index	[1,2,3,4, ...] [“hello”, “deaner”, ...]
Objects	Objects are special data types that have functions and data associated with them. These are more common in JS than PHP and we will need to use them often.	Document.getElementById(); (example of an object)
function	A user defined function that can be called by an user event (e.g. mouse click, etc)	function name () { statements; .. }

# Relation to computational thinking

26



```
var w = 10;  
var h = 10;  
var s = 0;
```

} Using the var keyword defines (or declares) the variables.

```
s = w + h;
```

} Once defined, we can refer to the variables by their name.

27

# Number variables & JS math

# Number type variables

28

```
var enrollment = -99;  
var medianGrade = 70.8;  
var credits = 5 + 4 + (2 * 3);
```

- Number types are integers (whole numbers) and numbers with decimal places
- Numbers with decimal places are often called "floating point" numbers, e.g.:

2.99993    3000.9999    -40.00

We call them floating point because the decimal point appears to float around. Sometimes these are just called *floats* to distinguish them from integers.

# Expressions and statements

29

- An expression is the combination of one or more variables, values, operators, or functions that computes a result.

```
var num1 = 5;                /* value 5 is the expression */

var num2 = num1 + 10;        /* num1 + 10 is the expression,
                             /* operator is +, this computes 5 + 10 */

num2 = num2 + 1;            /* this uses num2 and assigns the
                             result back to num */

var str1 = "hello";         /* value is "hello" */
var str2 = "world";        /* value is "world" */

num1 = ((3.14) * 10.0) / 180.0; /* multiple operators */
```

# Syntax breakdown of a statement

30

variable

```
num1 = ((3.14) * 10.0) / 180.0;
```

= is an assignment.  
This takes the result of the expression and makes it the current value of the variable on the left side of the assignment.

An expression that will be evaluated to compute some result. Later we will see this can be other things, like calling a "function", or testing if something is "true" or "false"

semicolon.  
In JS, we will use a semicolon to end most of our statements.

The entire entity above, that is the expression, assignment, and semicolon is called a "statement".

# Relation to computational thinking

31

In our computational thinking lecture, we wrote the following. This computes an expression using width and height and then assigns it to the variable p.

```
p ← 2*(width + height)
```

```
p = 2 * (width + height);
```

Assuming p, width and height have already been declared, this would be the corresponding JS code.

# Basic arithmetic operators

32

$a + b$	Addition	Sum of a and b.
$a - b$	Subtraction	Difference of a and b.
$a * b$	Multiplication	Product of a and b.
$a / b$	Division	Quotient of a and b.
$a \% b$	Modulo	Remainder of a divided by b.

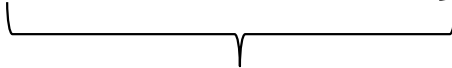
Here a and b could be variables, but we could also replace them with numbers.  $10 + 20$ ,  $3.14 / 2.0$ , etc. . .



# Evaluation and assignment

33

```
1: var num1 = 10;  
2: num1 = num1 + 10;
```



JS will interpret these statements as:

1: `num1` is assigned 10

2: `num1` + 10

10 + 10

20

`num1` ← 20

The expression *is always* computed **before** the assignment. This allows us to use a variable and assign the result back to the same variable.

# “Short hand” assignment operators

34

Assignment

`a += b;`

`a -= b;`

`a *= b;`

`a /= b;`

`a %= b;`

`a++;`

`a--;`

Same as:

`a = a + b;`

`a = a - b;`

`a = a * b;`

`a = a / b;`

`a = a % b;`

`a = a + 1;`

`a = a - 1;`

Addition

Subtraction

Multiplication

Division

Modulus

Self Addition

Self subtraction

# JS math operator precedence

35

```
var num1 = 5 * 5 + 4 + 1 / 2;    /* What is the answer? */  
var num2 = 5 * (5 + 4) + 1 / 2; /* What is the answer? */
```


*JS*

29.5 and 45.5

*output*

## Operator Precedence

( )                      Highest  
\* / %  
+ -                      Lowest



Compute results based on order of precedence, and from left to right in the expression.

\* This operator precedence is the same for most programming languages.

# Example from previous slide

36

$$\text{num1} = \underbrace{5 * 5}_{(a) \ 25} + 4 + \underbrace{1 / 2}_{(b) \ 0.5};$$
$$\underbrace{(a) \ 25 + 4}_{(c) \ 29}$$
$$\underbrace{(c) \ 29 + (b) \ 0.5}_{(d) \ 29.5}$$

Based on operator precedence, the expression would be computed in the following order:

(a)  $5 * 5 = 25$

(b)  $1 / 2 = 0.5$

(c)  $(a) + 4$  [where (a) is 25]

(d)  $29 + (b)$  [where (b) is 0.5]

final 29.5

$$\text{num1} = 5 * \underbrace{(5 + 4)}_{(a) \ 9} + \underbrace{1 / 2}_{(c) \ 0.5};$$
$$\underbrace{(b) \ 45}_{(d) \ 45.5}$$

Based on operator precedence, we would have:

(a)  $(5 + 4) = 9$

(b)  $5 * (a)$  [where (a) is 9]

(c)  $1 / 2 = 0.5$

(d)  $(2) + (c)$  [ $45 + 0.5$ ]

Final 45.5

37

# String variables

# String type variables

38

```
var s1 = "Connie Client";  
var s2 = 'Melvin Merchant';  
var favoriteFood = "falafel";
```

□ Strings are in treated like a series of characters

```
var favoriteFood = "falafel";
```

```
var stringNumber = "234";
```

Here, variable `stringNumber` is not the value two hundred and thirty four, but instead the characters 2,3,4.

# More on string type

39

```
var stringNumber = "234";  
var len = stringNumber.length; /* len is assigned the number 3 */
```

- String variables have a special property called "length" that returns the number of characters in the string.

```
var stringNumber = "234";  
var len = stringNumber.length;  
/* len is set to 3, since stringNumber  
   has 3 characters in it.          */
```

# String and spaces

40

```
var s1 = "String can have spaces"; // spaces are characters  
var len = s1.length;
```

- Keep in mind that spaces are also characters.
- That variable len will 22. Count the characters yourself -- make sure to count the spaces.



# Empty string

41

```
var s1 = ""; // This is something you will see  
var len = s1.length; // 0
```

- A variable can be assigned an empty string
- This means the variable is of type string, however, the string has no characters in it.
- The string will have length zero (0).

# String as an object

42

```
var s1 = "Connie Client";  
var len = s1.length;
```



variable `s1` is of type `String`, however, this can also be thought of as a "String object".

we can access various properties of the object using a "." operator and object's the property's name. You are going to see this type of property access often in JavaScript (and other "Object Oriented" languages)

# More on strings

43

- You need to put quotes around a string to let JS know it is a string. **You can also use single quotes.**

```
var s = "EECS1012"; // CORRECT!
```

```
var s = 'EECS1012'; // CORRECT!
```

```
var s = EECS1012; // INCORRECT! 
```

In this example, JavaScript will interpret EECS1012 as a variable, not a string!

# Special characters

44

- What if you want a quote character " to be part of the string?

```
var s = "This is a quote " "; //INCORRECT! ❌
```

This statement will cause problems for JS, because when it sees the second double quote it will assume this is the end of the string.

# Escape characters

45

- Escape characters are used inside strings to tell JS how to interpret certain characters

```
var s = "This is a quote \" "; //CORRECT
```

This string will be interpreted in JavaScript as:

T-h-i-s-\_-i-s-\_-a-\_-q-u-o-t-e-\_-"-\_-

Here a – is used to separated characters.

An underscore \_ is used to represent a space character.

# Alternatives

46

```
var answer = "It's alright";  
var answer = "He is called 'Johnny'";  
var answer = 'He is called "Johnny"';
```

You can use quotes inside a string, as long as they don't match the quotes surrounding the string

# More escape characters

47

Code	Result
<code>\"</code>	quote
<code>\'</code>	single quote
<code>\n</code>	New Line
<code>\\</code>	Backslash
<code>\t</code>	Horizontal Tabulator

Examples:

```
var x = 'It\'s alright.';
```

```
var x = "We are the so-called \"Vikings\" from the north.";
```

```
var x = "The character \\ is called backslash.";
```

```
var x = "This string ends with a new line \n";
```

# String concatenation (+ operator)

48

```
var s1 = "Hello ";  
var s2 = "World";  
var s3 = s1 + s3;      // s3 = "Hello World";
```

- The plus operator is used for string concatenation
- This can be confusing, because we often think of + as only being used for arithmetic. But in this case of String types, it means connect (or concatenate) two strings together.



# More string + examples

49

```
var s1 = "";           // empty string
var s2 = "Abdel";
var s3 = "Zhang";

var s4 = s1 + s2;      // result "Abdel", why? s1 is empty
var s5 = s2 + s3;      // result "AbdelZhang"
var s6 = s2 + " " + s3; // result "Abdel Zhang"
                        // why "Abdel" + " " + "Zhang" - adds a " "
s2 += s3;              // s2 now equals "AbdelZhang"
                        // why? s2+=s3; is the same as s2=s2+s3;
```

# String indexing [ ]

50

```
var name = "Deaner";  
var first_letter = name[0]; // assigns string "D"
```

*JS*

- We can access an individual character in the string using an index [ ] notation.
- The first character starts at index 0, not 1. This often confuses new programmers. See next slide.

# String indexing [ ]

51

```
var str1 = "J. Trudeau";
```

Index	0	1	2	3	4	5	6	7	8	9
Character	J	.		T	r	u	d	e	a	u

Expression	Result
<code>str1[0]</code>	"J"
<code>str1[3]</code>	"T"
<code>str1[2]</code>	" " (space character)
<code>str1.length</code>	10 (be careful - why 10?)
<code>str1[str1.length-1]</code>	"u"

We can think of a string as a sequence of characters. These characters can be “indexed” starting from zero (0).

# Unpacking last expression

52

`str1[str1.length-1]`

(1) get str1 length property (10)

`str1[ 10 - 1 ]`

(2) 10 - 1

`str1[ 9 ]`

(3) String index str[9]

`"u"`

(4) final result of expression

Our initial look at operator precedence didn't consider accessing objects.

## Operator Precedence

access object  
( ) parenthesis  
[ ]  
\* / %  
+ - .

Highest



Lowest

# String + number types

53

- When the + operator is used between variables or expressions that are string and numbers, **the number type will be converted to a string.**
- Examples:

```
var str1 = "how many? ";
var strNum = "10";
var num = 10;
str1 = str1 + num;           // result is "how many? 10".
strNum = num + strNum;      // result is "1010"
strNum = "" + num;         // result is "10"
```

This **last example** is a quick trick to convert any number type into a string. "" is an empty string. Adding a number to an empty string converts the number to a string.

54

# Arrays

# Arrays

55

## What is an array?

An array is a special variable, which can hold more than one value at a time. If you have a list of items (a list of car names, for example), storing the cars in an array would look like:

```
var car = ["Saab", "Volvo", "BMW" ];  
// We can access each individual value using the following notation.  
// car[0]    is "Saab"  
// car[1]    is "Volvo"  
// car[2]    is "BMW"
```

This is similar to how we can access individual characters in a String Type. Indexing starts from 0.

# Array properties

56

- Like Strings, arrays have a length property

```
var names = ["Amir", "Abdel", "Johan"];  
var len = names.length; // len is assigned 3
```



# Array can store different types

57

## Arrays can mix datatypes

```
var car = ["Saab", "Volvo", "BMW" ];    // Array of Strings
var nums = [ 1, 2, 3, 4, 5 ];          // Array of Numbers
var data = ["EECS1012", 780, "Fall", 2018 ]; // Mix types

// data[0] is a string type with value "EECS1012"
// data[1] is a number type with value 780
```

58

# Control Statements

# Control statements

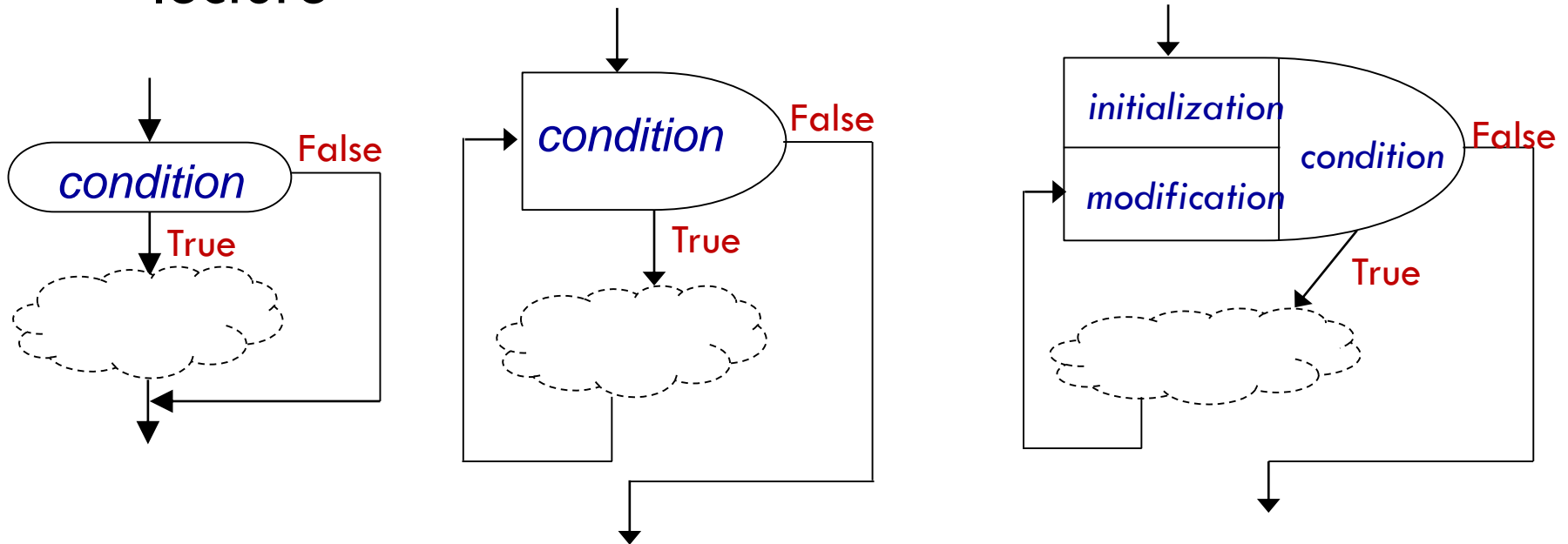
59

- Program flow control is the most powerful part of programming
- It allows us to make decisions on whether to execute some statements or not
- Virtually all programming languages have some type of control statements
  - The most basic are :
    - if statements
    - for or while statements (also called “loops”)

# Relation to computational thinking

60

- We saw flow control diagrams in our previous lecture

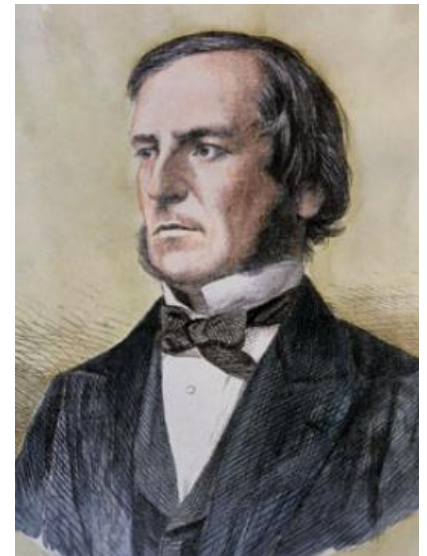


We will see how to implement these in JavaScript.

# Before we look at flow control

61

- It is important to understand basic Boolean logic and expressions
- Boolean logic concerns itself with expressions that are “true” or “false”
- The name comes from the inventor, George Boole



# True/false expressions

62

Expression	Meaning	Boolean Result
<code>45 &lt; 10</code>	is 45 less than 10? No.	FALSE
<code>45 &lt; 100</code>	is 45 less than 100? Yes.	TRUE
<code>50 &gt; -1</code>	is 50 greater than -1? Yes.	TRUE
<code>7 == 9</code>	Is 7 equal to 9? No.	FALSE
<code>8 == 8</code>	Is 8 equal to 8? Yes.	TRUE

Why the crazy double `==`? In JS, a single `=` sign means assignment. `var a = 5.`

So, to distinguish the assignment `=`, from the comparison if two items that are equal, we use a `==`.

# True/false example #2

63

This becomes more interesting when we use variables.

Expression	Meaning	Boolean Result
<pre>var a = 5; var b = 10;</pre>		
<code>a &lt; b</code>	is 5 less than 10? yes.	TRUE
<code>a == b</code>	is 5 equal to 10. no.	FALSE
<code>a &gt; b</code>	is 5 greater than 10? no	FALSE

# Equality with between Types

64

Expression	Meaning	Boolean Result
<pre>var a = 5; var b = "5";</pre>	Assignment as Number Assignment as String	
<pre>a == b</pre>	is Number 5 equal to string "5". In JS— yes!	TRUE
<pre>a === b</pre>	the triple-equal, tells JS to consider the type in the equality test. Is Number 5 equal to String "5" — no.	FALSE



# JS – comparison operators

65

Operator	Description	Comparing	Returns
==	equal to	<code>x == 8</code>	false
		<code>x == 5</code>	true
		<code>x == "5"</code>	true
===	equal value and equal type	<code>x === 5</code>	true
		<code>x === "5"</code>	false
!=	not equal	<code>x != 8</code>	true
!==	not equal value or not equal type	<code>x !== 5</code>	false
		<code>x !== "5"</code>	true
		<code>x !== 8</code>	true
>	greater than	<code>x &gt; 8</code>	false
<	less than	<code>x &lt; 8</code>	true
>=	greater than or equal to	<code>x &gt;= 8</code>	false
<=	less than or equal to	<code>x &lt;= 8</code>	true

# if statement

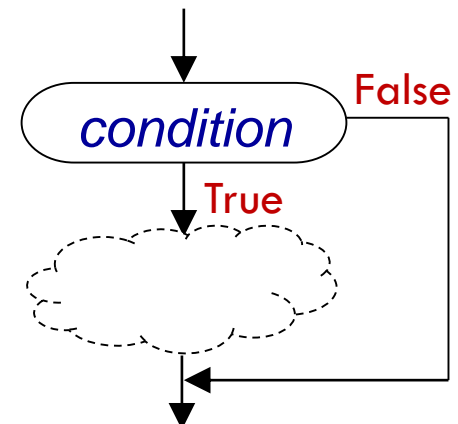
66

```
if (condition) {  
    statements;  
    ...  
}
```

JS

If statements execute code within the { } if the (**condition**) expression is true.

If the expression is false, the statements within the { } are skipped.



# If statement example

67

## Example

```
if (grade == "A")
{
    alert("I love EECS1012!");
}
```

*JS*

If the variable `grade` is equal to “A”, then the statements are executed. Otherwise the statements within the `{ ... }` are skipped.

# if/else statement

68

```
if (condition) {  
    statements1;  
} else {  
    statements2;  
}
```

JS

Almost the same as the if statement, but in this case, if the (**condition**) expression is false, *statements1* are skipped, but *statements2* are executed.

# if/else statement

69

## Example

```
if (grade == "A")
{
    alert("I love EECS1012!");
}
else
{
    alert("I hate EECS1012!");
}
```

JS

If the variable grade is equal to "A", then the statements are executed. Otherwise the statements within the else { ... } are executed.

# while-loops

70

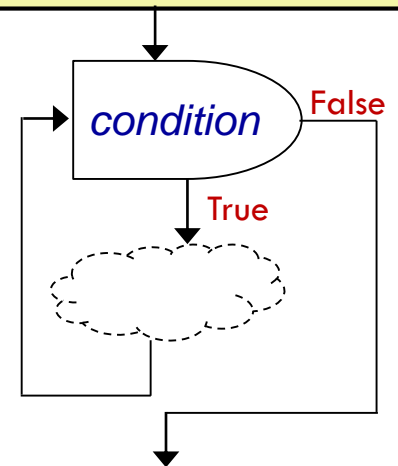
```
while (condition) { // while the condition is true
    statements;
}
```

JS

Example

```
var i=0;
var sum = 0;
while (i < 100) { /* loop i is less than 100 is true */
    sum = sum + i; /* adds up 0 to 99 */
    i++;          /* adds one to i */
}
```

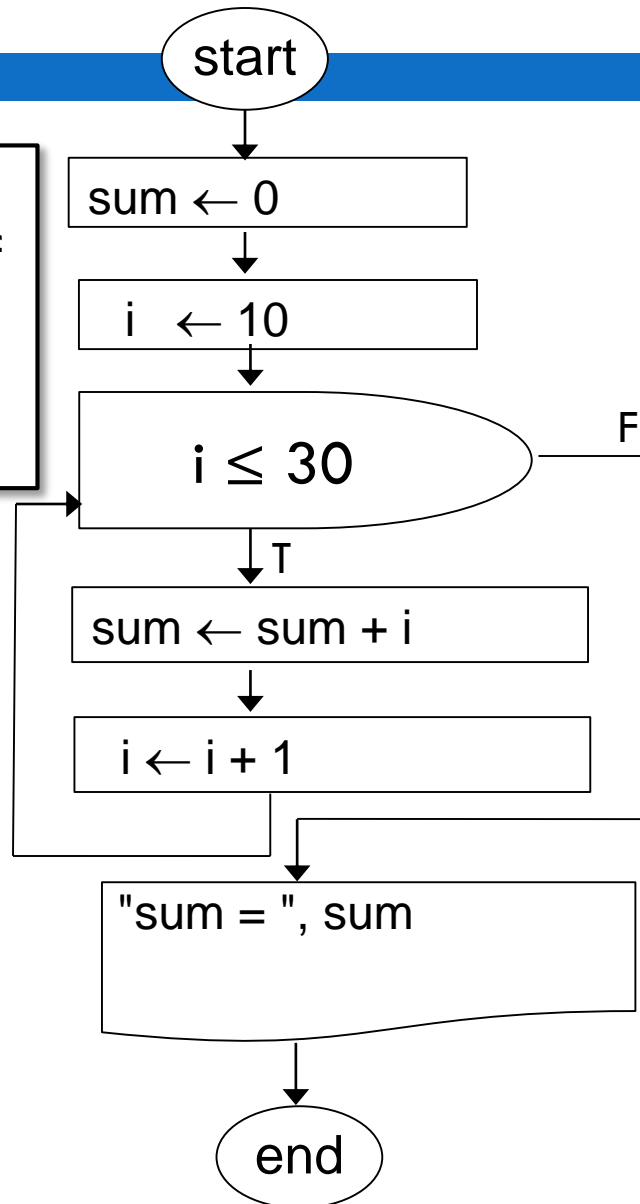
JS



# Computational thinking example

71

**TASK**  
compute the sum of numbers between 10 and 30, inclusively.



```
var sum = 0;
var i=10;
while (i <= 30) {
  sum = sum + i;
  i = i + 1;
}
alert("sum = "+sum);
```

JS

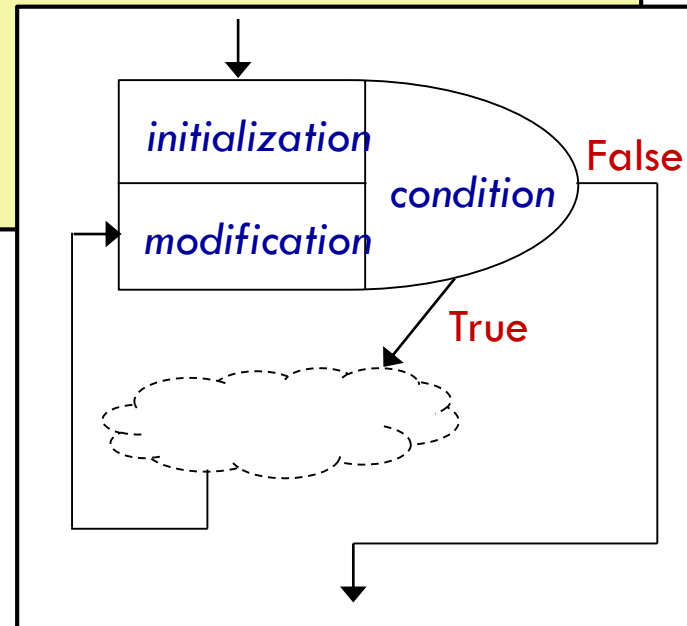
# for-loop

72

```
for (intitialization; condition; update) {  
    statements;  
    ....  
}
```

JS

```
var s1 = "hello";  
var s2 = ""; /* empty string */  
for (var i = 0; i < s.length; i++) {  
    s2 += s1[i] + s1[i];  
}  
// s2 will equal "hheellllloo"
```



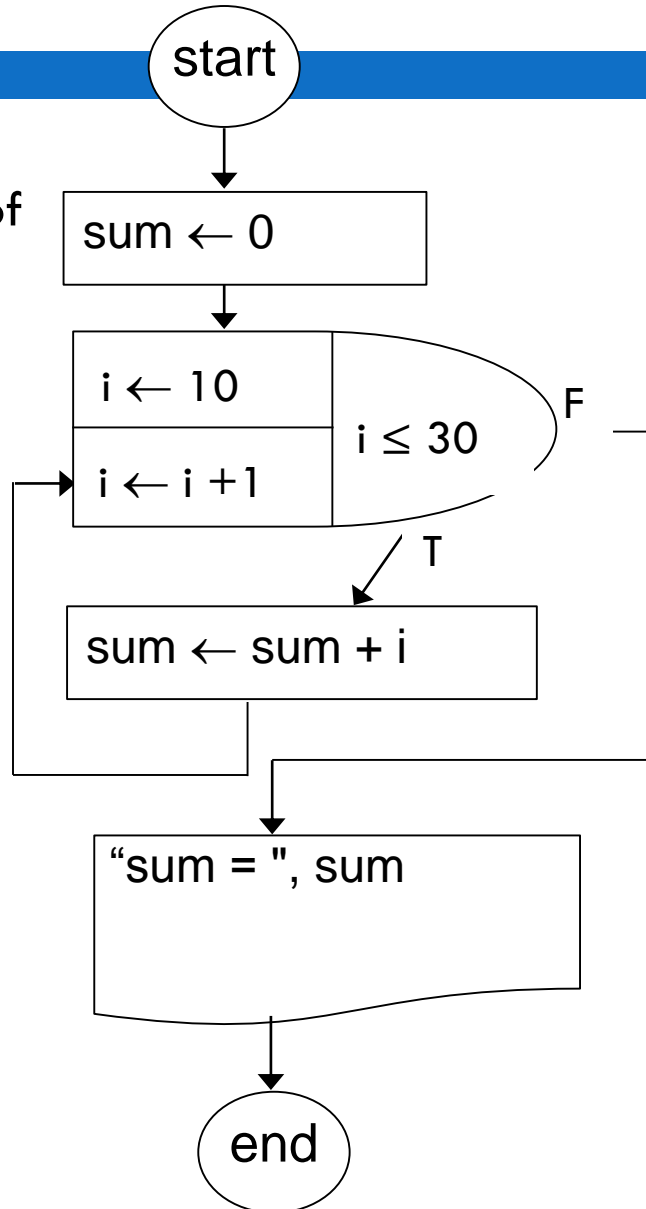


# Computational thinking example

73

start

compute the sum of numbers between 10 and 30, inclusively.



```
var sum=0;
for(var i=10; i<= 30; i++)
{
    sum = sum + 1;
}
alert("sum = "+sum);
```

JS

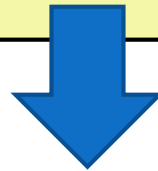
# for-loop more detail

For-loops are special cases of while-loops. They allow a fast way to specify the initialization, condition, and update rules for a while loop.

74

```
for (initialization; condition; update) {  
    for_statements;  
}
```

JS



for-loop logic as a while loop

```
initialization statement;  
while (condition expression)  
{  
  
    for_statements;  
  
    update statement; //apply after for statements  
}
```

75

# JavaScript Functions

# JavaScript - functions

76

- We saw these on slide 14, this section gives more details
- A JavaScript function is a block of code that performs some task
- A function is executed when something "calls" or "invokes" the function

# Function syntax – several examples

77

1)

```
function name() {  
  statements;  
  ...  
}
```

Keyword `function` is used to define a function.

`name` is the name of the function.

The parenthesis are used to denote it is a function that accepts no parameters. See next slide.

2)

```
function name(parameter1, parameter2, ...) {  
  statements;  
  ...  
}
```

This syntax allows parameters to be "passed" to the function. Parameter names are defined between the `(..)` (see slide 71)

3)

```
function name(parameter1, parameter2, ...) {  
  statements;  
  ...  
  return value;  
}
```

This syntax allows parameters to be "passed" to the function.

Note that the function **also** returns a value.

(see slide 72)

# Function: Ex 1

78

```
<html>
<head>
  <script src="example.js" type="text/javascript"></script>
</head>
<body>
<button onclick="myFunction();">Click me!</button>
</body>
</html>
```

## example.js

```
function myFunction() {
    alert("You clicked my function!");
}
```

Simple function that calls an alert box with the text "You clicked my function!".

# Function: Ex 2

This example combines many concepts. Functions, parameters, arrays, and if-else statements.

79

```
<html>
<head>
  <script src="example.js"
type="text/javascript"></script>
</head>
<body>
<button onclick="myFunction(2);">Click me!</button>
</body>
</html>
```

## example.js

```
function myFunction( num ) {
  var words = ["zero", "one", "two"];
  if (num < words.length)
  {
    alert(words[num]);
  } else
  {
    alert("more than two");
  }
}
```

The parameter name is **num**. **num** is a variable that takes the value that was placed when the function was called.

If the **num** is 2 or less, then print out the word in the array. Otherwise, print out "more than two".

# Function: Ex 3

80

```
<html>
<head>
  <script src="example.js"
type="text/javascript"></script>
</head>
<body>
<button onclick="myFunction(2);">Click me!</button>
</body>
</html>
```

## example.js

```
function doubleVar( p ) {
  var result = p + p;
  return result;
}

function myFunction( num ) {
  var doubleValue= doubleVar( num );
  alert( "Double " + doubleValue );
}
```

The JS file define two functions. The first, named `doubleVar()` takes a single parameter, named `p`. It computes `p+p` and returns it.

The second, named, `myFunction()`, takes a parameter, named `num`. The value in `num` is passed to the first function. The returned result is assigned to variable `doubleValue`. This is displayed in an alert box.



81

# JavaScript Objects

# Objects

82

- Number and string variables are containers for data
- Objects are similar, but can contain many values.
- Objects also can associated functions (they are called methods to distinguish them from the functions you just learned about).
- We will examine several pre-defined Objects in JavaScript

# Math Object

83

```
/* PI is value associated with the Math object. We access it
using the "." operator, just like we did with length for arrays
and strings. num now equals 3.14159265358979 */
var num1 = Math.PI;

var num2 = -50.30;
var num3 = 4;
var num4 = 66.84

var result1 = Math.round(4.7); // method rounds a number
var result2 = Math.abs( num2 ); // method computes absolute value
var result3 = Math.sqrt( num3 ); // method computes the square root
var result4 = Math.min( num2, num3 ); // returns the minimum of a list of nums
var result5 = Math.max(num2, num3); // returns the maximum of list of nums
var result6 = Math.floor(num4); // rounds number down to nearest integer
var result7 = Math.ceil(num4); // rounds up to nearest integer
```

# Useful Math methods

84

Function	Description
<code>Math.abs(n)</code>	absolute value
<code>Math.ceil(n)</code>	ceil means round up to the nearest integer 9.01 would round up to 10.
<code>Math.floor(n)</code>	floor means round down to the nearest integer 9.99 would round down to 9.
<code>Math.min(n1, n2, ..)</code> , <code>Math.max(n1, n2, ..)</code>	min or max of a sequence of numbers: e.g. <code>max(50, 43, 1, -1, 30) = 50</code>
<code>Math.sqrt(n)</code>	computes square root of n
<code>Math.random()</code>	return a random number between 0 (included) and 1 (excluded). So, the number will be between 0 and 0.999999999...
<code>Math.round(n)</code>	Traditional round approach, e.g. 9.4999 would round to 9; 9.50 would round up to 10.

# Math Object

85

## □ Random

- Random is a useful *Math* object method that generates a random floating point number between 0 (inclusive) and 1 (exclusive)

```
// returns a number between 0 - 1. 0 is included, but not 1.  
var num1 = Math.random();
```

```
// returns a number between 0 - 99  
var result = Math.floor(Math.random() * 100);
```

```
// returns a number between 0 - 100  
var result = Math.floor(Math.random() * 101);
```

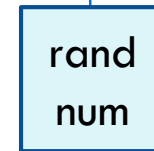
# Explaining previous examples

86

```
var result = Math.floor( Math.random() * 100);
```

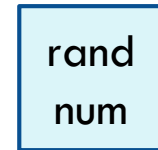
1 **Math.random()** is called first.

Returns a number between 0-.9999999.. (remember 1 is not included).



2 **Then multiple by \* 100.**

That will result in a number between 0 - 99.999999..



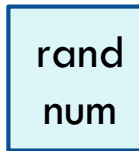
\* 100

3 **Floor() is called last.**

Applying the floor() method rounds *down to the nearest integer*. So now the range is 0 .. 99.



Math.floor(



\* 100 )

# Date object

87

- Date object allows us to get information about the date.
- The format is different than the Math object. In this case, we need to use the "new" keyword to create a new Date Object which is assigned to a variable.

```
var myDate = new Date();

var day = myDate.getDay();           // returns day of the week
var year = myDate.getFullYear();     // returns the year
var month = myDate.getMonth();       // returns the month
var minute = myDate.getMinutes();    // returns the minute
var second = myDate.getSeconds();    // returns the seconds
var dateStr = myDate.toString();     // returns a string of the date
```

# Date methods

88

Method	Description
<code>getDate()</code>	Returns the day of the month (from 1-31)
<code>getDay()</code>	Returns the day of the week (from 0-6)
<code>getFullYear()</code>	Returns the year (e.g. 2018)
<code>getHours()</code>	Returns the hour (from 0-23)
<code>getMilliseconds()</code>	Returns the milliseconds (from 0-999)
<code>getMinutes()</code>	Returns the minutes (from 0-59)
<code>getMonth()</code>	Returns the month (from 0-11)
<code>getSeconds()</code>	Returns the seconds (from 0-59)



# document Object

- The document object is another useful built-in object in JavaScript. We will learn more about this in detail in upcoming lectures.
- Here, we will show how to use the document object to change the text inside a paragraph.

# document Object

90

```
function myFunction() {  
    var p = document.getElementById("mydata");  
    p.innerHTML = "You clicked the button!";  
    // this changes the text of the HTML for this object.  
    // later we will see this isn't the best way, but  
    // will help you get started.  
}
```

`document.getElementById("id name")`

Object "document".  
Note that the object  
name is **lowercase!**

EECS1012

Call the object's method `getElementById(...)`  
searches the HTML page to find the element  
with the `id=="id name"`.

If the element isn't found, the methods returns  
**null**.

# Example using document object (1)

91

```
<!DOCTYPE html>
<html>
<head>
  <script src="example.js"
type="text/javascript"></script>
</head>
<body>
  <p id="mydata"> button not clicked </p>
  <button onclick="myFunction();" > Click Me!
</button>
</body>
</html>
```

Event (click) of the HTML button,  
calls the specified handler function

JS file: [example.js](#)

```
function myFunction() {
  var p = document.getElementById("mydata");
  p.innerHTML = "You clicked the button!";
}
```

# Example using document object (2)

92

```
function myFunction() {  
    var p = document.getElementById("mydata"); // get the paragraph  
    p.innerHTML = "You clicked the button!"; // changes HTML code  
}
```

<p id="mydata"> →

button not clicked

Click Me!

you clicked the button

Click Me!

After myFunction() called.

93

# Putting it all together examples

# You have the basics to get started

94

- From the previous slide, you have the basics to get started
- We will show a few examples in the next few slides
- These are also posted on the "Additional Resources" on the class schedule page

# HTML file

95

```
<!DOCTYPE html>
<html lang="en">
<head>
  <!-- link to external JS file. Note that <script> has an
  end </script> tag -->
  <meta charset="utf-8">
  <title> Example 2 </title>
  <script src="example2.js" type="text/javascript"></script>
</head>
<body>
  <!-- Create a paragraph with id mydata -->
  <p id="mydata"> Button no clicked yet. </p>
  <button onclick="myFunction();"> Click Me! </button>
</body>
</html>
```

Button not clicked yet.

Click Me!

# Example 2 – random number

96

```
function myFunction()
{
  var num = Math.random();          // get a random number
  var p = document.getElementById("mydata");    // get the paragraph

  if (num < 0.5)                    // if num less than 0.5
  {
    p.innerHTML = num + " is less than 0.5 ";
  }
  else
  {
    p.innerHTML = num + " is equal to or large than 0.5";
  }
}
```



# Example 2 – random number (2)

97

Button not clicked yet.

Click Me!

Before any click.

0.11516930158266092 is less than 0.5

Click Me!

First click calls function.  
HTML of paragraph is changed.

0.578331354153119 is equal to or large than 0.5

Click Me!

Other clicks also calls function.  
HTML of paragraph is changed.

# Example 3 – Random greeting

98

```
/* A function that returns a random number between 0 and 3 - see slide 86 */
function myRandom() { /
    var num = Math.floor( Math.random() * 4 );
    return num;
}

/* function called my our HTML page when the button is clicked */
function myFunction() {
    var greetings = ["Hello", "Yo", "Hi", "Welcome"]; // declare array
    var selectOne = myRandom(); // get random number between 0 -3
    var p = document.getElementById("mydata"); // get paragraph
    p.innerHTML = greetings[ selectOne ]; // set paragraph
}
```

# Example 3 – random number (2)

99

Button not clicked yet.

Click Me!

Each click generates a new random number and outputs the corresponding greeting in the array.

Welcome

Click Me!

Hi

Click Me!

Yo

Click Me!

# Example 4 – for loops and string +

100

```
<p id="mydata"> Button no clicked yet. </p>
<button onclick="myFunction(15);"> Click Me! </button>
```

```
/* called when button is clicked. Passes a value from the HTML page */
function myFunction(num)
{
    var sum = 0;
    var outputString = "Adding 0"
    var p = document.getElementById("mydata");

    for(var i=1; i <= num; i++)
    {
        sum = sum + i;
        outputString = outputString + "+" + i;
    }
    p.innerHTML = outputString + "= " + sum;
}
```

Adding 0+ 1+ 2+ 3+ 4+ 5+ 6+ 7+ 8+ 9+ 10+ 11+ 12+ 13+ 14+ 15= 120

Click Me!

# Example 5 – Date, array, if

101

```
/* function called by button click */
function myFunction()
{
  var p = document.getElementById("mydata");
  var dayNames = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"];
  var myDate = new Date();
  var day = myDate.getDay();
  // recall day is 0=sunday, monday=1, and so on
  if (day == 2) // check if the day is Tuesday
  {
    p.innerHTML = "Today is Tuesday!"; // if it is
  }
  else {
    p.innerHTML = "Today is NOT Tuesday!"; // if it is not
  }
  // finally
  p.innerHTML += "<br> Today is " + dayNames[day];
}
```

Today is NOT Tuesday!  
Today is Sun

Click Me!

# Comments on notation

102

In many programming languages, you will see the following notations

**value**

- Text by itself is assumed to be a **variable** or **object** named *value*

**"value"**

- Text with quotes is assumed to be a **string** with content *value*

**value()**

- Text with parentheses after is assumed to be a **function** name *value* or a *method associated with an object*.

# Summary

103

- You have learned the basic of JavaScript
- We will look at more details to the document object next
- We will also see how to allow JavaScript to be applied to a page without the need for the HTML to be modified (e.g. with no `onclick=""` properties).