ARCH A4988

Coding for Spatial Practices

<JavaScript Basics>

Learning Objectives

Learning Objectives

Giving your website some interactivity

1. Describe the role JavaScript plays alongside HTML and CSS.
2. Define website behavior and the practical uses of JavaScript.
3. Demonstrate the ability to include JavaScript files in a project.

Agenda

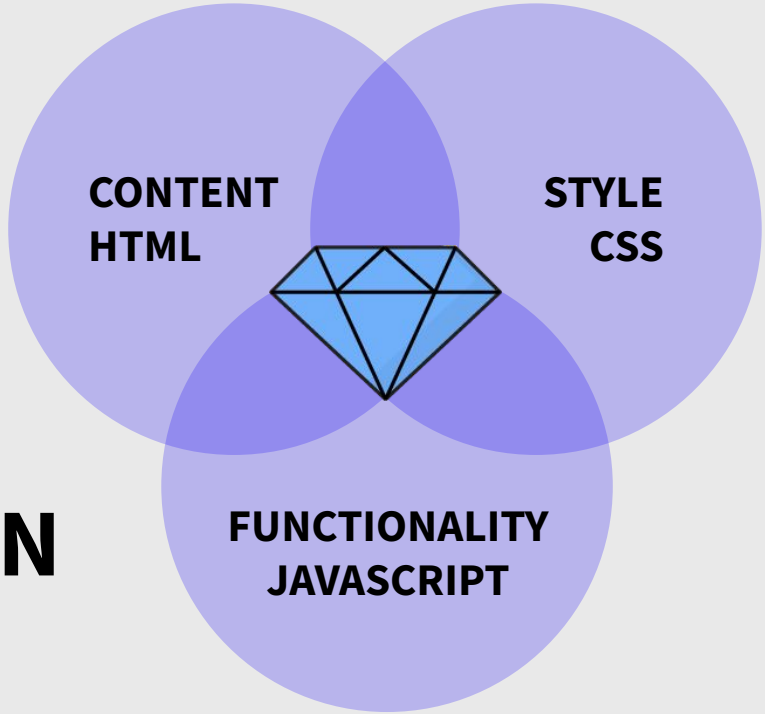
1. Hello, World!
2. JavaScript Basics
3. Exercises
4. Final Project -- Deliverable 02

What is JavaScript?


JAVASCRIPT
HTML
CSS

Front End Trifecta

BONES



SKIN

BRAIN

JavaScript Basics

What is JavaScript?

A list of instructions to be executed by a computer:

- Instruction-based
- Conditional
- Sequential

JavaScript Basics

What JavaScript can do?

- Access the content of the page
- Modify the content of the page
- Program rules or instructions the browser can follow
- React to events triggered by the user or browser
- Use an API via HTTP(S)

JavaScript Basics

What JavaScript can do?

1. Access Content
2. Modify Content
3. Program Rules
4. React to Events

You can use JavaScript to select any element, attribute or text from an HTML page.

Example:

- Find out what a visitor to your site entered into a text input when they submit a form

JavaScript Basics

What JavaScript can do?

1. Access Content
2. Modify Content
3. Program Rules
4. React to Events

You can use JavaScript to add (or remove) elements, attributes and text to/from an HTML page.

Example:

- Change the size, position, color or other styles for an element

JavaScript Basics

What JavaScript can do?

1. Access Content
2. Modify Content
3. Program Rules
4. React to Events

You can use JavaScript to specify a set of steps (instructions) for the browser to follow.

Example:

- Have images/text fade in if the user has scrolled to a certain portion of the page

JavaScript Basics

What JavaScript can do?

1. Access Content
2. Modify Content
3. Program Rules
4. React to Events

You can use JavaScript to specify that a script should run when an event occurs.

Example:

- When a button is clicked
- When the cursor hovers over an element

How to Implement

Internal

```
<script language="javascript" type="text/javascript">
JavaScript code
</script>
```

External

```
<script language="javascript"
src="js/filename.js"></script>
```

Hello World

```
<script language="javascript" type="text/javascript">
    alert("Hello World!")
</script>
```

Hello World

Method

```
<script language="javascript" type="text/javascript">
    alert("Hello World!")
</script>
```

Hello World

Bracket

```
<script language="javascript" type="text/javascript">
    alert("Hello World!")
</script>
```

```
() data
[] an array
{} object or the opening of a block statement
```

Hello World

Data

```
<script language="javascript" type="text/javascript">
    alert("Hello World!")
</script>
```

JavaScript Basics

Whitespace & Line Breaks

```
<script language="javascript"
type="text/javascript">
    let num01 = 10
    let num02 = 20
</script>
```

Semicolons

```
<script language="javascript"
type="text/javascript">
    let num01 = 10; let num02 = 20;
</script>
```

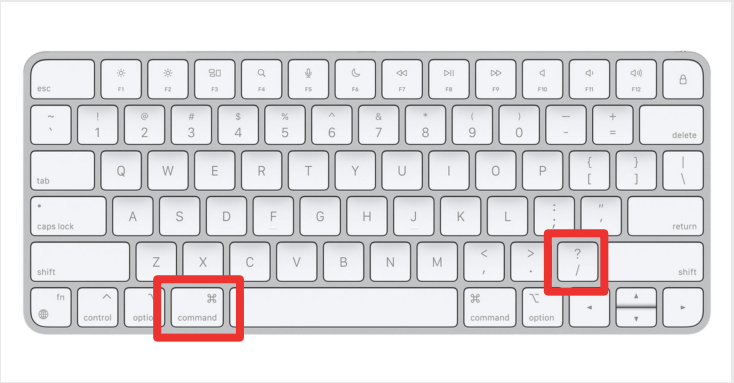
JavaScript Basics

Comments

```
<script language="javascript" type="text/javascript">
  // This is a single line comment
  <!-- This is a multiline comment
  -->
  /* This is also a multiline comment */
</script>
```

Shortcut

Command + backslash



JavaScript Basics

console

```
<script language="javascript" type="text/javascript">
    console.log("Hello World!");
</script>
```

JavaScript Basics

document

```
<script language="javascript" type="text/javascript">
    document.write("Hello World!");
</script>
```

JavaScript Basics

Variables

Sometimes your program will have to store temporary bits of information it needs to do its job. It uses variables to store this information.

Before you can use a variable, you need to announce that you want to use it. This involves creating it and giving it a name.

```
var width;
```

JavaScript Basics

Variables

Once you've created the variable, you can tell it what information you would like it to store for you.

```
width = 40;
```

JavaScript Basics

Variables

Before the newer version of JavaScript was introduced in 2015, var was used extensively. Now, with JavaScript ES6 we use `let` and `const`.

JavaScript Basics

Variables

Declare variables using the `let` or `const` keywords.

You refer to variables by using their names anywhere in your program.

```
let age;
age = 29;
let age = 29

let numberOfStudents = 12;
const name = "Celeste";
```

JavaScript Basics

Variable Naming

Names should be easily understood.

- **No spaces allowed**
- You can use "_"; but don't do it
- Use camelCase: itWorksLikeThis

```
let howManyCoasters = 18;
const midWeek = "Wednesday";
```

JavaScript Basics

Variable Reassignment

Note: If you'd used **const** instead of **let** to declare the variable, the last part of the example would throw an error!

```
let day = "Tuesday";
day = "Wednesday";

let x = 18;
console.log(x);
⇒ 18

x * 2;
console.log(x);
⇒ 18

x = x * 2
console.log(x);
⇒ 36
```

JavaScript Basics

Methods

The actions that can be performed on objects.

alert(); console.log(); click(), toggle(), mouseover(), mousemove(), mouseenter(), mouseleave(), drag(), drop(), scroll(), show(), hide(), fadeIn(), fadeOut(), slideDown(), slideUp(), focus(), blur(), css(), addClass(), html(), append(), load(), copy(), paste(), ...

JavaScript Basics

Function

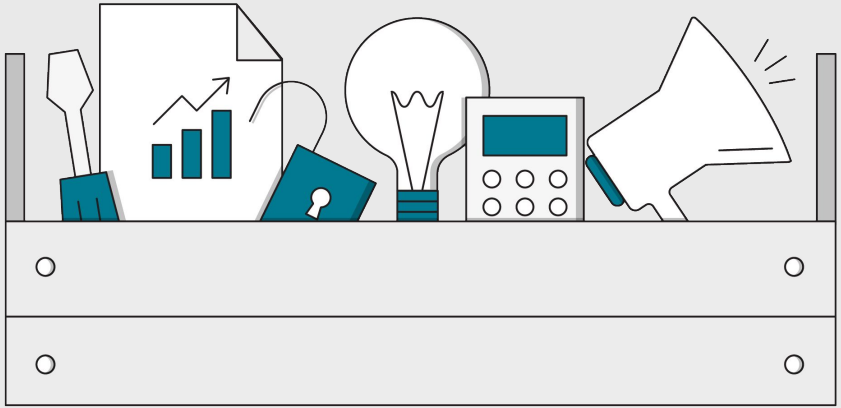
A block of code designed to perform a particular task.

```
//define function called sayHello
function sayHello() {
  console.log("Hi!");
}

//call sayHello Function
sayHello();
```

JavaScript Basics

Basic Data Types

- `Strings`
- `Numbers`
- `Boolean`
- `Null / undefined`



JavaScript Basics

Numbers

- You've probably seen these before: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- They are used mathematically throughout JS code, so normal math rules apply.
- They are paired with operators: `+`, `-`, `*`, and `/`.
  - `10 * 10 ⇒ 100`
  - `8 - 4 ⇒ 4`
  - `49 / 7 ⇒ 7`
- They can also include floating point numbers — i.e., decimals or floats.
  - `8.99 + 2 => 10.99`

JavaScript Basics

Strings

- String means **text** — that's it!
  - `"It is a beautiful evening";`
  - `"Is it really Monday?";`
  - `"I am feeling good today!";`
- With JS, you can merge strings using the "**+**" operator. This is called "**string concatenation**."
  - `"You want " + "to go " + "eat on 14th?";`
- You can think of a string as a collection of **characters** tied together.

JavaScript Basics

Boolean

- Booleans represent the logical concept of **true or false**.
  - Other data values can be converted to Booleans for logical analysis.
  - **`0`**, **`-0`**, **`null`**, **`NaN`**, **`undefined`**, or the empty string (**`""`**) are **`false`**.
  - All other values will be converted to **`true`** — if it exists, it's "truthy."

```
Boolean("Harry Styles");
⇒ true

Boolean("1979");
⇒ true
```

JavaScript Basics

Null / undefined / NaN

- These values denote the lack of value in JavaScript.
  - **`null`** specifically suggests nothing — i.e., certain not to be anything.
  - **`undefined`** suggests a variable will be given a value later but not yet.
  - **`NaN`** means “not a number,” usually because your math has gone wrong.

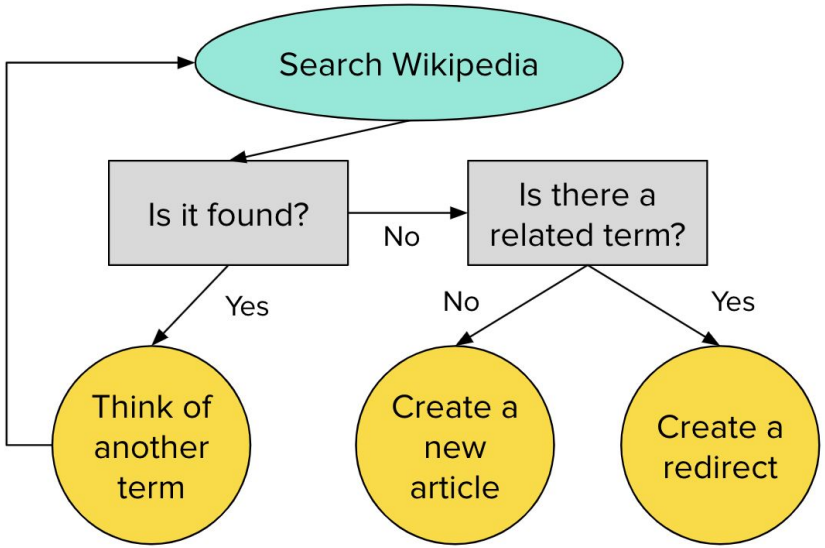
```
let lysine;
console.log(lysine);
⇒ undefined

console.log(9 * null);
```

JavaScript Basics

Conditional Statements

Why use them?

Conditional logic allows you to create vastly more complex programs. Think of the decision-making process of giant flowcharts.



Conditional Statements

Comparison Operators

These are a set of operators that give you the ability to compare values and return a Boolean result (true or false).

operator	name	description	example
>	Greater than	Returns true if the left operand is greater than the right operand.	a > b
>=	Greater than or equal	Returns true if the left operand is greater than or equal to the right operand.	b >= a
<	Less than	Returns true if the left operand is less than the right operand.	a < b
<=	Less than or equal	Returns true if the left operand is less than or equal to the right operand.	b <= 5

Conditional Statements

Equality Operators

operator	name	description	example
==	Equal	Returns true if the operands match. Does not compare data type.	3 == var1
!=	Not equal	Returns true if the operands are not equal. Does not compare data type.	var1 != 4
===	Strict equal	Returns true if the operands are strictly including their data types.	3 === var1
!==	Strict not equal	Returns true if the operands are not equal and/or not of the same type.	var2 !== 3

Double vs. Triple Equals

== vs ===

!= vs !==

Double equals doesn't check for type, so it won't care if the data types are the same. This can lead to some unpredictable behavior!

Triple equals — the strict equality operator — will compare both the value and data type, creating much more predictable results. **Just use triple equals!**

Conditional Statements

Conditionals are function-like statements that take Booleans as inputs:

```
if (valueOne === valueTwo) {
  console.log(“Valid”);
}
```

Note the **code block** defined in curly braces (`{ }`). This code block executes if the Boolean provided is **true**.

`else` Statements

You will often want to have an `else` statement immediately after the `if` statement. This will trigger when the `if` comparison turns out to be `false`.

```
if (valueOne === valueTwo) {
  console.log(“Valid”);
} else {
  console.log(“Invalid”);
}
```

Multiple Conditions

```
if (test1 === test2) {
  console.log('test1 and test2 are equal');
} else if (test1 > test2) {
  console.log('test1 is greater than test2');
} else {
  console.log('none of the conditions were met');
}
```

Careful! Equals Aren't All Equal...

When you use =, that is an **assignment operator**.

If you try to use = instead of === in a comparison statement, you will get strange results — it will always evaluate as **`true`**!

```
// This won't work the way you think it will!
let temperature = 0;
if (temperature = 100) {
  console.log("Always going to happen!");
}
```

Conditional Statements

Logic Operators

These allow us to combine multiple conditions together. For very complex conditionals, you can put several conditions in parentheses to evaluate them as a single expression.

operator	name	description	usage
&&	Logical AND	Evaluates to **`true`** only if all combined values are true.	expr1 && expr2
||	Logical OR	Evaluates to **`true`** if any of the combined values are true.	expr1 || expr2
!	Logical NOT	Reverses the Boolean result of whatever follows it.	!expr

Multiple Conditions, One Statement

You can check to see if two conditions are met in one statement with a logic operator:

```
let clickDetected = true;
let clickTwoDetected = false;

if (clickDetected === true && clickTwoDetected === false){
  // do something
}
```

Using !

The **not** operator, `!`, is a great way to check if something exists in the JS memory system.:

```
let whatever; // would be undefined, no memory assignment

if( !whatever ) {
  console.log(“Turns out whatever doesn’t exist!”);
}
```

If there isn’t a comparison inside a conditional statement, JS will interpret the argument as a Boolean. The only values that would equate to false are `0`, `-0`, `null`, `NaN`, `undefined`, or `“”`, so nearly anything that exists will pass the test.

Pro Tip: Condensing Conditionals

Like we saw in the previous slide, not all conditionals need a comparison statement — especially if the values being tested are already Booleans!

Thus, you will rarely see a comparison with `=== true` or `!== false`.

Instead, developers will use the following pattern:

```
if (clickDetected && !clickTwoDetected) {
  // do something
}
```

JavaScript Basics

Arithmetic Operators

operator	name	description	usage
-	Negation	Subtracts	4 - 3 = 1
+	Plus	Adds	4 + 3 = 7
*	Multiply	Multiplies	3 * 2 = 6
/	Divide	Divides	12 / 2 = 6
%	Modulus	Returns the remainder.	12 % 5 = 2
++	Increment	Increases the value by 1	x = 1; x++; x = 2
--	Decrement	Decreases the value by 1	x = 1; x--; x = 0

Arrays

Think of arrays as containers of data. Technically, an **array** is an ordered collection of data types combined into one variable.

Each item in an array is assigned an **index** value based on its position. These index values allow us to access individual elements within the array.

```
["banana", "orange", "apple"]
```

0 **1** **2**

Why Do Arrays Matter?

Arrays are one of just two types of data “containers” in JavaScript (the other is objects). As a result, much of the data we use comes in the form of arrays.

Arrays come from three main places:

1. The DOM (with `querySelectorAll` and `getElementsByClassName`)
2. API responses (information received from other web applications)
3. Databases

Syntax

```
const fruits = ["banana", "orange", "apple"]
```

You make an array using a set of square brackets.

Inside the brackets, each value must be separated by a comma.

Accessing Array Values

```
const fruits = ["banana", "orange", "apple"];
fruits[0]; // will output "banana"
fruits[1]; // will output "orange"
fruits[2]; // will output "apple"
```

Access array items using square brackets around their index values. It’s pretty simple — just remember that the first index value is always zero!

.length

```
const fruits = ["banana", "orange", "apple"];

fruits.length;
=> 3
```

Use the `.length` property to figure out how many items are in your array. This is very useful when we need to look through the whole array with a loop.

Removing Items With .pop

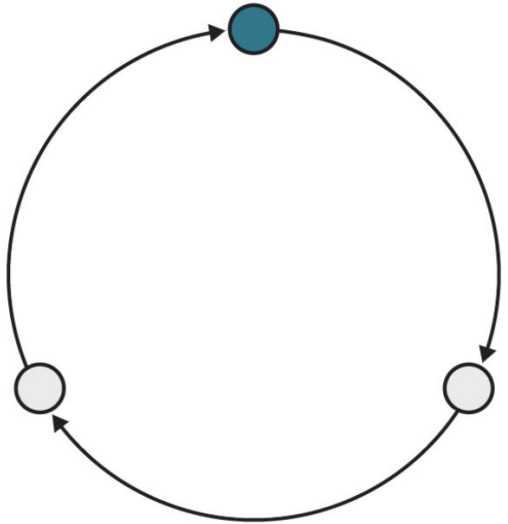
```
const fruits = ["banana", "orange", "apple"];

fruits.pop();
// fruits is now: ["banana", "orange"];
```

The .pop method removes the last item in the array. Can you hear the sound effect when you pop an item off?

Loop: A control flow statement allowing for the repeated execution of a code block until a specific condition is reached.

Why Loops?

- **Loops** take advantage of what computers do best: *evaluate instructions across organized sets of data very quickly*.
- Computers excel when working in isolated patterns, which is exactly how a loop works.
- Avoid needlessly copying or re-typing code by repeating it in a loop.



An Iterator, Terminator, and Incrementer Walk Into a Loop...

A **`for` loop** is similar to an `if` statement but with more conditions. When creating a `for` loop, we need to make three declarations:

1. Define a variable to act as our **iterator**, typically named `i`.
2. Establish a condition for the loop to stop, called the terminating condition.
3. Increment the iterator variable (or decrement, if the loop goes backward).

```
for (let i = 0; i < 10; i++) {
  console.log(i);
}
// outputs 0,1,2,3,4,5,6,7,8,9
```

An Iterator, Terminator, and Incrementer Walk Into a Loop...

A **`for` loop** is similar to an `if` statement but with more conditions. When creating a `for` loop, we need to make three declarations:

1. Define a variable to act as our **iterator**, typically named `i`.
2. Establish a condition for the loop to stop, called the terminating condition.
3. Increment the iterator variable (or decrement, if the loop goes backward).

```
for (let i = 0; i < 10; i++) {
  if(i === 6){ break; }
  console.log('Total elephants: ' + i);
}
```

The Structure of a for...loop

The **`for` loop** consists of three optional expressions, followed by a code block:

1. Initialization -- a used to create a counter.
2. Condition -- checked each time before the loop runs. If it evaluates to true, the statement or code in the loop is executed. If it evaluates to false, the loop stops.
3. Final expression -- executed after each iteration of the loop; usually used to increment or decrement a counter

```
for (initialization; condition; finalExpression) {
  // code
}
```

The Structure of a for...loop

Iterators

```
const students = ["Alice", "Han", "Chi Chi", "Brent"];

for(let index = 0; index < students.length; index++){
  console.log("Name of student is: " + students[index]);
}
```

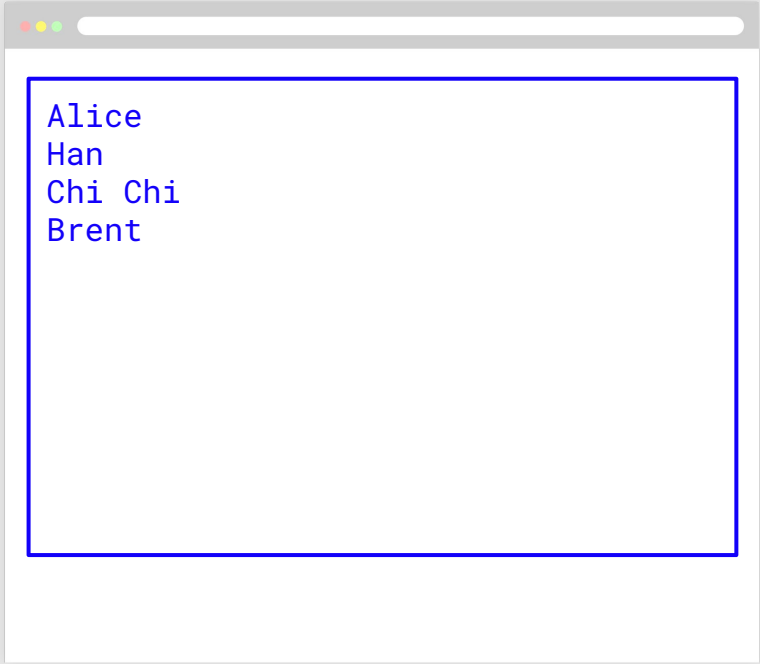
The Structure of a for...loop

Iterators

```
const students = ["Alice", "Han", "Chi Chi", "Brent"];


                  1           4           1+1
for(let index = 0; index < students.length; index++){
  students[1] // Han
  console.log("Name" + students[index]);
}
```

The Structure of a for...loop

Iterators



Guided Walk-Through:

GSAPP Event Page

20 minutes

Let’s practice writing functions.

Instructions:

Go to the Google Drive course folder, **Week-09: JavaScript Basics** and download **starter_code_week_09** into your course folder. Open ***arrays_and_loops***

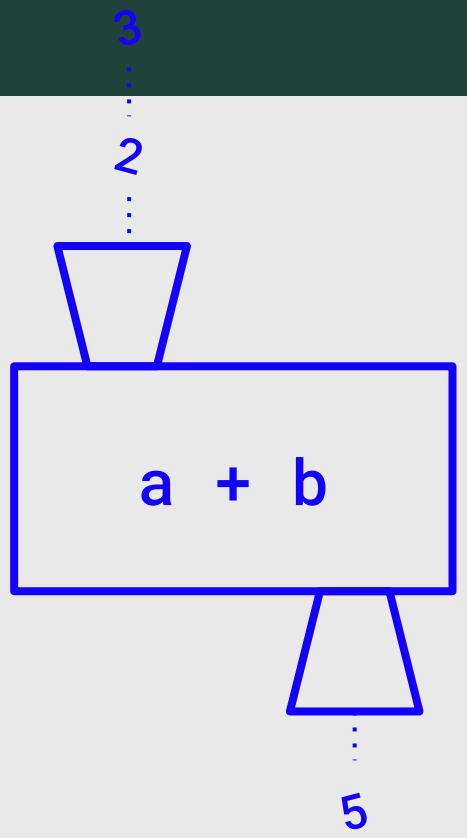
Functions

Function Basics

What is a function?

Function is a term that comes out of mathematics. A function is a block of code that returns a result.

It only knows what you tell it and when you ask it something, it will reply with an answer.



Function Basics

Why use functions?

1. Group Steps
2. Reusability
3. Store Steps

Function Basics

Why use functions?

1. Group Steps
2. Reusability
3. Store Steps

Functions allow you to group a series of statements together to perform a specific task

Function Basics

Why use functions?

1. Group Steps
2. Reusability
3. Store Steps

We can use the same function multiple times. This saves you time from writing more code.

Function Basics

Why use functions?

1. Group Steps
2. Reusability
3. Store Steps

Functions are not always executed when a page loads. It provides us with a way to ‘store’ the steps needed to achieve a task.

Anatomy of a Function

keyword

function name

scope

function body

return statement

semicolon

```
function calculateBill(){
    let sum = 3 + 2;
    return sum;
};

calculateBill();
```

Anatomy of a Function

keyword

function name

scope

function body

return statement

semicolon

```
function calculateBill(){
   let sum = 3 + 2;
   return sum;
};

calculateBill();
```

function name + parentheses

Anatomy of a Function

keyword

function name

scope

function body

return statement

semicolon

```
function calculateBill(){
   let sum = 3 + 2;
   return sum;
};

calculateBill();
```

scope start

scope end

Anatomy of a Function

keyword

function name

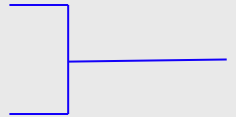
scope

function body

return statement

semicolon

```
function calculateBill(){
    let sum = 3 + 2;
    return sum;
};

calculateBill();
```



Anatomy of a Function

keyword

function name

scope

function body

return statement

semicolon

```
function calculateBill(){
   let sum = 3 + 2;
   return sum;
};

calculateBill();
```

Recognizing the parts of a Function

Function Container

```
function multiply() {

};
```

Recognizing the parts of a Function

Input parameters

```
function multiply(num1, num2) {
  // now you have two variables you can access
  // num1 and num2
};
```

Recognizing the parts of a Function

Input parameters

```
function multiply(num1, num2) {
  // now you have two variables you can access
  // num1 and num2
};
```

Recognizing the parts of a Function

Output

```
function multiply(num1, num2) {
  // Output
  return num1 * num2
};
```

Guided Walk-Through:

GSAPP Event Page

20 minutes

Let’s practice writing functions.

Instructions:

Go to the Google Drive course folder, **Week-09: JavaScript Basics** and download **starter_code_week_09** into your course folder. Open ***functions_javascript***

Function Syntax

Call a function

To run the code in a function, we call, or invoke, the function by using the function name followed by parentheses.

If the parentheses are not included, the function will not run.

```
function sayHello(){
  console.log(“Great to see you.”)
};

sayHello();
```

Function Syntax

Properly name a function

```
checkInputLength()
moveSquareRight()
changeBackgroundColor()
```

The variable you use for a function should contain a verb. If the purpose of your function is to check data, for example, use the verb check in the variable name.

Functions usually do things:

- getting data
- setting data
- checking data
- printing data

Functions with Parameters

Write a function with one parameter

In the function, the parameter is arbitrarily called *name*. We call parameters whatever makes semantic sense.

```
// Define the function
function sayName(name){
  console.log('Hello! My name is
' + name);
}

// Call (or invoke) the function
with a parameter
sayName('Helen')
sayName('Mabel')
```

Functions with Parameters

Write a function with one parameter

Instead of logging the statement to the console, what if we render it to the webpage?

```
// Define the function
function sayName(name){

  const body = document.querySelector('body');

  const p = document.createElement('p');

  p.textContent = ‘Hello! My name is‘ + name

  body.appendChild(p)
}

// Call the function with a parameter
sayName(‘Helen’)
sayName(‘Mabel’)
```

Functions with Parameters

Write a function with multiple parameters

A function can take any number of parameters.

```
// Define the function
function calculateArea(width, length){
  console.log(width * length);
}

// Call (or invoke) the function with a parameter
calculateArea(4, 4)
calculateArea(12, 16)
```

Functions with Parameters

Write a function with a return statement

A function is only defined if it has a return value.

```
// Define the function
function calculateArea(width, length){
  return width * length;
}

// Call (or invoke) the function with a parameter
calculateArea(4, 4)
calculateArea(12, 16)
```

`return` — available to use in another function

Functions with Parameters

Stop a function

You can use return to terminate a function.

If the **num** value is present, the function goes on as expected, otherwise it's immediately stopped.

```
function calculateSomething(num) {
  if (!num) {
    return
  }

  // go on with the function
}
```

return

Programs with multiple, reusable functions that process and `return` data are the next step in your journey.

```
function addThings(val1, val2) {
  return val1 + val2;
}

let result = addThings(1, 2);
console.log(result);
```

return + Conditions

Functions will often have conditions that change the nature of their output.

```
function addThings(val1, val2) {
  if (val1 >= 10) {
    return val1 + val2;
  }
  return 0;
}

console.log(addThings(11, 2));
console.log(addThings(7, 5));
```

Practical Application

These examples may seem kind of silly (they are!), but that's because our data is static.

Loops and functions become more important when we handle **live data** from APIs and databases, because they help us deal with information we haven't defined ourselves!

Guided Walk-Through:

GSAPP Event Page

20 minutes

Image: *Products for Fluxus editions*, 1964.
Digital Image © The Museum of Modern Art/Licensed by SCALA / Art Resource, NY

Let's make an event score in the style of the Fluxus visual artist, Alison Knowles. The Fluxus movement emphasized process over product.

Instructions:

Go to the Google Drive course folder, **Week-08: JavaScript Basics** and download **starter_code_week_08** into your course folder. Open ***house_of_dust***

Exercise

Class Exercise

Part I

Using your new knowledge of some simple JavaScript functionality, to solve the following:

Imagine you work the information booth at a theme park and help recommend rides to guests.

1. Declare a variable age. Assign it the value 25.
2. Declare a variable height. Assign it the value 5.
3. Log each variable to the console and hit the "Run" button in the console panel. Example: console.log(age)

Class Exercise

Part II

Write out an if / else if / else statement for the following conditions:

1. If a person is less than 8 years old, recommend the merry-go-round.
   `console.log("Check out the Merry-Go-Round. You'll love it!");`
2. Otherwise if a person is more than 8 years old AND less than 65 years old AND more than 4.5 feet tall, recommend the roller coaster.
   `console.log("Check out the Roller Coaster. It's awesome!");`
3. Otherwise recommend the lazy river
   `console.log('Why not enjoy a float down the Lazy River?');`

Class Exercise

Sentence Generator

1. Create three variables called noun verb and adjective and store one of each type.
2. Choose a short one sentence poem that includes the following variables:
   - Sample sentence: `My ${noun} leaps ${adjective} when I ${verb} a rainbow in the sky:`
3. Create five different versions of this sentence with different variables.
4. Style the HTML pages.

Class Exercise

Sentence Generator

1. Make Make a list of at least five words for each variable:
   – Sample array: let nouns = [‘heart’, ‘rainbow’, ‘ocean’];.
2. Create a randomly generated sentence by using the variables.
   – Sample sentence: `My ${noun} leaps ${adjective} when I ${verb} a rainbow in the sky:`
3. Style the HTML page.

Hint: Formula for selecting a random element from an array

```
let item = arrayName[Math.floor(Math.random()*arrayName.length)];
```

Upload all your html files to your Github repository, and add links to the pages on the Github homepage.

Project 03 – Catalog!!!

Project 03

Create a collection of 100 (visual) media items. These can be physical objects that you document, screenshots, found images, videos, original images, etc. If you want to collect quotes or text, it will have to be displayed with an image-based method.

We will add these into a database to be determined, *(you have the option to swap collections with one of your peers)*. We will then make an online experience that connects with the database in order to learn how to pull structured data that is not yours and use it to populate a website.

Project 03

Deliverable 01 << 10/29/2024 >>

1. Come up with 3 ideas for a collection with 5 images/media items/examples for each. (example set of three themes: screenshots of my desktop, video clips from the news, or cut out images of doll body parts).
    a. Consider what unites the media, how different or similar can they be from each other?
    b. You don’t have to know why you like something or where this is going!
2. For each, write a summary of what you’d like to investigate with the collection idea.

Project 03

Objectives:

- Find a way for the type of content to inform the form of your site
- To gather and organize a collection of media
- To design flexibly for content that we can’t control
- To connect and use structured content from an “API” that you created

Project 03

Consider:

- WHAT do you want to do and WHY, and then HOW?
- Does the type of content imply a form for your site?
- How can you tell a story through a curated set of visuals / interactive experiences?
- How is the collection organized (or disorganized)?
- Consider how the user interacts with this collection, do they see it all at once, or do they have to work to uncover parts or all of it?
- Does the design of the site respond to new content? (e.g. marking anything that has been added in the last 24 hours)

Project 03

Requirements:

- A title for your collection
- An about page or section containing a brief text about your collection + designer credit
- Must include all 100+ media items
- Your site must have one javascript element that enhances the experience and relates to your collection in a meaningful way.
- Your website should have at least two ways of viewing your collection.
- These can be filters/subgroupings, sorting methods, list v thumbnails toggle, etc.
- Site favicon
- Must be responsive / functional on a mobile screen (mobile does NOT need a custom design, however, but the site should not break on a mobile screen.)

Project 03

Deliverable 02 << 11/12/2024 >>

1. Create a folder that bears your name in the Google Drive folder: Final-Project-Student-Folders. Come up with three layout ideas for your website.
2. Consider ways that you might organize the 100 items. Come up with multiple ideas for a set of 3 sub-categories. These might be filtering or sorting mechanisms. Find similarities, draw connections, and develop a concept or narrative for the collection you chose.

Project 03

Deliverable 02 << 11/12/2024 >>

3. Present sketches of 3 vastly different ways of viewing/designing your assigned collection using Sketch, XD, Figma, or Indd.
    - Can you look at one item at a time, or is the overall-ness important?
    - How does the user move between the items in your collection?
    - How do the filters/sorting mechanisms function?
4. Add the sketches to your Google Drive project folder: Final-Project-Student-Folders.