Before we begin...

- Open up these slides:
  - https://goo.gl/zzs1Uo

Data Types, Loops & Functions

Learning Objectives

- **Talk** about the common looping structures
- **Identify** potential use cases of loops
- **Identify** and explain composite data types
- **Define** what an array is in JavaScript
- **Define** what an object is in JavaScript
- **Use** arrays and objects effectively in JavaScript
- **Identify** the need for functions in JavaScript
- **Use** functions effectively, including providing data as *parameters*/*arguments*, and *returning* information

Agenda

- Loops
- Composite Data Types
  - Arrays
  - Objects
- Iteration
- *Functions*
  - *Parameters/arguments*
  - *Return values*

A quick review

- Primitive Data Types
- Variables
- Conditionals
- Comparison Operators
- Logical Operators

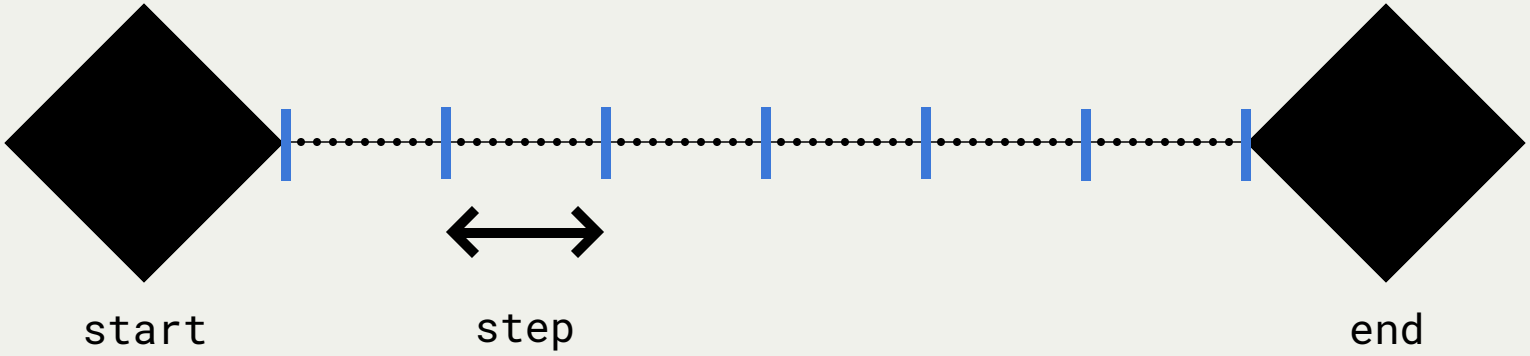
Loops

What are loops?

What are loops?

- A piece of code that can execute over and over again
- Loops are made up of three main parts:
  - A **starting point**
  - An **increment**, or **step**
  - An **ending point**
  - You always need these things!
- You get access to the current value
- We can use JS functionality in a loop (e.g. run conditionals) and you can stop loops (using the break statement)

Start, Step, End



The *while* loop

The *while* loop

```
while ( condition ) {
    // Statement(s) to repeat
}
```

```
var count = 0;

while ( count < 5 ) {
    console.log( count );
    count = count + 1;
}
```

The *for* loop

The *for* loop

```
for ( start; end; step ) {
    // Statement(s) to execute
}
```

```
for (var i = 0; i <= 10; i += 1) {
    console.log( i );
}
```

Some advice...

I would stick to the *for* loop in the beginning, because:

- It will make sure you have all the necessary parts (start, end, step)
- You can see all the important things right at the beginning of the statement

Exercise

Do the exercises found here

Composite Data Types

What are they?

Composite Data Types?

- Composite Data Types are types that are built from other types
- More complex than primitive data types
- Think of them as:
  - Data structures
  - Data with distinguishable parts

What composite types do we have in JavaScript?

Composite Data Types

In JavaScript, we have two main composite data types:

1. **Arrays**
    - Ordered and you access data with an *index*
2. **Objects**
    - Unordered and you access data with a *key*

Arrays

What are arrays?

- They are lists that can be filled with any data type
  - Both primitive and composite
- Ordered and you access data with an *index*
  - An index is a number and it is zero-based (meaning the first item is always 0)
- They are able to be *iterated* through (meaning looped through)
- Think of them as todo lists

Creating Arrays

```
var emptyArray = [];

var randomNumbers = [ 12, 42, 1, 3, 92 ];

var dataTypes = [ true, null, 14, "string" ];

var weirdInstruments = [
    "Badgermin",
    "The Great Stalacpipe Organ",
    "Stylophone",
    "Ondes Martenot",
    "Sharpischord",
    "Hydraulophone",
    "Pyrophone"
];
```

Accessing Elements

```
var weirdInstruments = [
    "Badgermin",
    "The Great Stalacpipe Organ",
    "Stylophone",
    "Ondes Martenot",
    "Sharpischord",
    "Hydraulophone",
    "Pyrophone"
];

weirdInstruments[0];
weirdInstruments[5];
weirdInstruments[ weirdInstruments.length - 1 ];
```

Reassigning Elements

```
var weirdInstruments = [
    "Badgermin",
    "The Great Stalacpipe Organ",
    "Stylophone",
    "Ondes Martenot",
    "Sharpischord",
    "Hydraulophone",
    "Pyrophone"
];

weirdInstruments[0] = "Roli Seaboard";
weirdInstruments[5] = "Makey Makey Banana Piano";
weirdInstruments[ weirdInstruments.length - 1 ] = "OP1";
```

Looping through Arrays

```
var ordinals = [
    "Zeroth",
    "First",
    "Second",
    "Third"
];

ordinals[0];
ordinals[1];
ordinals[2];
ordinals[3];

// Fair bit of consistency there!
```

Looping through Arrays

```
var ordinals = [
    "Zeroth",
    "First",
    "Second",
    "Third"
];

for ( var index = 0; index <= 3; index += 1 ) {
    var currentElement = ordinals[index];
    console.log( currentElement );
}
```

Looping through Arrays

```
var ordinals = [
    "Zeroth",
    "First",
    "Second",
    "Third"
];

for ( var index = 0; index <= ordinals.length; index += 1 ) {
    var currentElement = ordinals[index];
    console.log( currentElement );
}
```

Properties & Methods

```
var ordinals = [
    "First",
    "Second",
    "Third"
];

ordinals.length; // => 3

ordinals.pop(); // Remove the last element
ordinals.push( "Third" ); // Add "Third" to the end

ordinals.shift(); // Remove the first element
ordinals.unshift( "First" ); // Add "First" to the start

ordinals.indexOf( "Second" ); // Get the index of "Second" => 1
```

Properties & Methods

Lots of others! Some of the more common ones:

- `.join`
- `.slice`
- `.includes`
- `.reverse`
- `.forEach`
- `.reduce`
- `.filter`
- `.map`

Exercises

Do the exercises found here

Resources

- CodeAcademy
- Array Documentation
- Speaking Javascript: Arrays
- Eloquent Javascript: Arrays
- Javascript.info's Description

Objects

What are objects?

- Objects are unordered
- They are similar to dictionaries
- They are a collection of **key-value pairs** (like a word and a definition in a dictionary)
- They can store any data types

Why use objects?

- Encapsulation and modularity
  - Ways to group functionality
  - Makes sharing your code easier (e.g. as a library)
- Give names to values
- Work with large amounts of data effectively

Creating Objects

```
var emptyObject = {};

var movie = {
    name: "Satantango",
    director: "Bela Tarr",
    duration: 432
};
```

Creating Objects

```
var bookSeries = {
    name: "In Search of Lost Time",
    author: "Marcel Proust",
    books: [
        "Swann's Way",
        "In the Shadow of Young Girls in Flower",
        "The Guermantes Way",
        "Sodom and Gomorrah",
        "The Prisoner",
        "The Fugitive",
        "Time Regained"
    ]
};
```

Accessing Values

```
var movie = {
    name: "Satantango",
    director: "Bela Tarr",
    duration: 432
};

var movieName = movie.name;
var movieDirector = movie.director;
var movieDuration = movie.duration;

// OR...
var movieName = movie["name"];
var movieDirector = movie["director"];
var movieDuration = movie["duration"];
```

Changing Values

```
var movie = {
    name: "Satantango",
    director: "Bela Tarr",
    duration: 432
};

movie.name = "Sátántangó";
movie.director = "Béla Tarr";
movie.duration = 534;
```

Adding new Values

```
var movie = {
    name: "Satantango",
    director: "Bela Tarr",
    duration: 432
};

movie.language = "Hungarian";
movie.rating = 2141252324616982; // Out of 5
movie.parts = 12;
```

Nested Objects

```
var explorer = {
    firstName: "Jacques",
    lastName: "Cousteau",
    birth: {
        day: 11,
        month: 6,
        year: 1910
    }
};

var birthDay = explorer.birth.day;
var birthMonth = explorer.birth.month;
var birthYear = explorer.birth.year;
```

Complex Data Structures

```
var marxFamily = [
    { name: "Groucho", birthYear: 1890 },
    { name: "Harpo", birthYear: 1888 },
    { name: "Chico", birthYear: 1887 },
    { name: "Zeppo", birthYear: 1901 },
    { name: "Gummo", birthYear: 1893 }
];

for ( var i = 0; i < marxFamily.length; i++ ) {
    var brother = marxFamily[ i ];
    console.log( brother.name, brother.birthYear );
}
```

Exercise

Do the exercises found here

Resources

- Sitepoint
- Speaking JavaScript
- Eloquent JavaScript
- Code Academy
- JavaScript.info

Functions

What are functions?

- A reusable section of code that has a purpose and a name
- The bread and butter of JS
- They can associate names with subprograms

What are functions?

Creating new words is normally bad practice, though fun.
It is essential in programming!

We give a name to a part of our program, and in doing so, we make it flexible, reusable and more readable

How do they work?

- We **define** a function
- We **call** (or **execute**) it when we want the code within the function to run

What can functions do?

They can perform any code!

- Calculations
- Animations
- Change CSS
- Change, add, or delete elements on the page
- Speak to a server (e.g. an API)
- Anything!

Declaring Functions

```
// A Function Declaration

function sayHello () {
    console.log( "Hello" );
}

// A Function Expression

var sayHi = function () {
    console.log( "Hi" );
};
```

Calling Functions

```
function sayHello () {
    console.log( "Hello!" );
}

sayHello(); // The callsite
```

Calling Functions

```
var sayHello = function () {
    console.log( "Hello!" );
}

sayHello(); // The callsite
```

Parameters || Arguments

They aren't dynamic... yet! This brings us to parameters or arguments

Parameters (or **arguments**) allow us to provide a function with extra data or information. This is what makes a function flexible!

Parameters || Arguments

```
function sayHello ( name ) {
    var greeting = "Hello " + name;
    console.log( greeting );
}

sayHello( "Groucho" );

sayHello( "Harpo" );

sayHello(); // ???
```

Parameters || Arguments

```
function multiply (x, y) {
    console.log( x * y );
}

multiply( 5, 4 );

multiply( 10, -2 );

multiply( 100, 0.12 );
```

Some Pseudocode

```
changeTheme function
    RECEIVE a themeChoice ("light" or "dark")
    IF themeChoice === "light"
        CHANGE the body background to "white"
        CHANGE the text color to "black"
    ELSE
        CHANGE the body background to "black"
        CHANGE the text color to "white"

moveToLeft function
    RECEIVE an element to animate
    STORE the current left position as currentLeft
    STORE the new left position, as desiredLeft, by adding 100px to currentLeft
    UPDATE the left position of the provided element to be desiredLeft
```

Return Values

Sometimes your function calculates something and you want the result!

Return values allow us to do that

We can store the result of calculations with return values. Think of **.toUpperCase();**

Return Values

```
function squareNumber ( x ) {
    var square = x * x;
    return square;
};

var squareOfFour = squareNumber( 4 );
```

Return Values

```
function squareNumber ( x ) {
    var square = x * x;
    return square;
};

var squareOfFour = squareNumber( 4 );

var squareOfTwelve = squareNumber( 12 );

squareNumber(8) + squareNumber(11);

squareOfFour + squareOfTwelve;
```

Return Values

```
function square ( x ) {
    return x * x;
};

function double ( x ) {
    return x * 2;
}

var result = double( square( 5 ) );
```

Return Values

```
var userOne = {
    admin: true
};

var userTwo = {
    admin: false
};

function isAdmin (user) {
    var admin = user.admin;
    return admin;
}

isAdmin(userOne); // true
isAdmin(userTwo); // false
```

Return Values

```
function sayHello () {
  return "No.";
  console.log( "Hi!" );
};

sayHello();
```

- A **return** value means that a function has a result
- It is always the last line that executes

Passing in Variables

```
var addTwoNumbers = function (x, y) {
  return x + y;
};

var firstNumber = 10;

addTwoNumbers( firstNumber, 4 );
addTwoNumbers( firstNumber, 6 );
```

Callbacks

```
function runCallback ( cb ) {
    // Wait a second
    cb();
}

function delayedFunction () {
    console.log("I was delayed");
}

runCallback( delayedFunction );
```

Function Guidelines

Follow the F.I.R.S.T principle:

- **F**ocussed
- **I**ndependent
- **R**eusable
- **S**mall
- **T**estable

*Also, make it error-tolerant. But that isn't in the acronym

Exercise

Do the exercises found here

Resources

- Function Documentation
- Speaking Javascript: Functions
- Eloquent Javascript: Functions
- Javascript.info's Description

Homework

- Finish all exercises from class
  - Loops
  - Arrays
  - Objects
  - Functions
- Upload your homework to GitHub
- Prepare for next lesson

Homework (Extra)

- Go through The Modern JavaScript Tutorial
- Read Eloquent JavaScript
- Read Speaking JavaScript

What's next?

- Functions
  - Parameters/arguments
  - Callbacks
  - Return values
- Scope in JavaScript

Questions?

Feedback

https://ga.co/js05syd

Thanks!