Before we begin...

- Open up these slides:
  - https://goo.gl/57KZmN

Data Types, Loops & Conditionals

Learning Objectives

- **Identify** and explain primitive data types
- **Define** what a variable is in JavaScript
- **Distinguish** between static and dynamic typing
- **Explain** the execution of a program
- **Structure** a program using control flow statements
- **Use** comparison and logical operators effectively
- **Talk** about the common looping structures
- **Identify** potential use cases of loops

Agenda

- Primitive Data Types
- Variables
- Conditionals
- Operators
- Loops

A quick review

Data Types

What are they?

What are they?

- Data types are the types of information provided by a programming language
- **Primitive** data types are the basic building blocks of a language
  - They are *immutable* (cannot be changed)
- There are also **composite** types. These are types of data that can be constructed by combining primitive and other composite data types
  - They are *mutable* (can be changed)

What types do we have in JavaScript?

Primitive Data Types

Data Type	Description	Example
Strings	Pieces of text	"hello world"; 'a'
Numbers	Integers or floats	4; 2.6;
Booleans	Represents true or false	true; false;
Undefined	Has not been assigned a value	undefined;
Null	Non-existent	null;
*Symbol**	*Anonymous data*	*Symbol("hello");*

* Symbols were added in ES6

Composite Data Types

Data Type	Description	Example
Objects*	Collection of data	{ name: "Jane" }
Arrays	Ordered collection of data	4; 2.6;
Functions	A "callable" subprogram	function hi () {}

* Technically, Objects are the only true composite data type

Strings

What are strings?

A sequence of characters used to represent text

Delimited by single or double quotes*

Special characters have to be escaped (e.g. conflicting quotes)

Strings

```
"";

"A piece of text";

'These are all strings';

"Jane's bag";

'Bill\'s guitar';

"Hello " + "World"; // Concatenation
```

Properties & Methods

```
// Properties

"Hello World".length;

// Methods

"Hello World".toUpperCase();

"Hello World".startsWith("H");
```

Numbers

What are numbers in JS?

- Integers
- Floats (Decimals)
- Negative
- Positive
- Some weird things...

Numbers

```
42;

1294921;

0.14;

-14;

-19.2521;
```

Arithmetic Operators

```
4 + 8;   // Addition

12 - 4;  // Subtraction

4 * 5;   // Multiplication

8 / 4;   // Division

9 % 4;   // Remainder
```

Comparison Operators

```
9 === 9; // Strict Equality

4 > 3;   // Greater than

6 >= 2;  // Greater than or equal to

6 < 19;  // Less than

3 <= 3;  // Less than or equal to


// What do these return?
```

Booleans

What are booleans?

A logical data type that can only have the value **true** or **false**

They are used primarily to determine which sections of code to execute (e.g. the *if* conditional) or which sections of code to repeat (e.g. the *for* loop)

Booleans

```
true;

false;
```

Undefined

What is undefined?

A data type meaning that there is no value. This often pops up when:

- A *variable* has just been declared and hasn't been assigned a value
- You attempt to access an *object's property* and it has no associated value
- You receive a *parameter* in a *function*, but no *argument* has been provided for that *parameter*

This is what the browser sets automatically

Undefined

```
undefined;
```

Null

What is null?

A null value represents having no data stored

It, just like undefined, means there is no value

The difference?

- Null is often intentional, or explicit (e.g. you store this value in a variable)
- Undefined is set by the browser (e.g. when you don't give a variable a value)

Null

```
null;
```

Variables

What are they?

What are variables?

- Named containers for storing, and accessing, data
- Variables are ways to store information in memory
  - You assign a name to a certain piece of data or collection of code. Think of them as named buckets

How do we create them?

Three steps

1. **Declaration**
    - Registers a variable in the *scope*
2. **Initialization**
    - Allocates memory for the variable (*automatic in JS*)
3. **Assignment**
    - Assigns a specified value to the variable

Ways of creating variables

To create a variable:

- The **var** *keyword*
- A JavaScript *identifier* (the name of the variable)
- A single equals sign (not like maths! This means *assignment* in JS)
- A *value*

```
var myVariable = "Some data";
```

Ways of creating variables

To create a variable:

- The **var** *keyword*
- A JavaScript *identifier* (the name of the variable)

```
var myVariable;
```

What value would this variable be given?

Naming Conventions

Naming Conventions

The JavaScript identifier:

- Must be one word
- Must consist only of letters, numbers, the dollar sign and underscores
- Must **not** start with a number
- Must **not** be one of these reserved words
- Is case sensitive

Camel-casing

When you create a variable, pick a name based on clarity and meaning. Be descriptive!

If you need to use two or more words in your JS identifier, use camel-casing! This is where you capitalize all words after the first word

```
var camelCasing = true;
var dataTypes = true;
var playerOneMove = "Rock";
```

Typing

Vars can change types

```
var myVariable = true;

myVariable = "a string";

myVariable = 42;

myVariable = null;

myVariable = undefined;
```

The above is all fine! Note that I don't use the var keyword after the first time. This is because the variable has already been *declared*!

Static vs. Loose Typing

Static vs. Loose Typing

- Static typing (also called strong typing, or strict typing)
  - You have to declare what type a variable is going to be (e.g. this variable is always going to be a string)
- Loose typing (also called weak typing, or dynamic typing)
  - The data stored in a variable can change

What typing does JavaScript have?

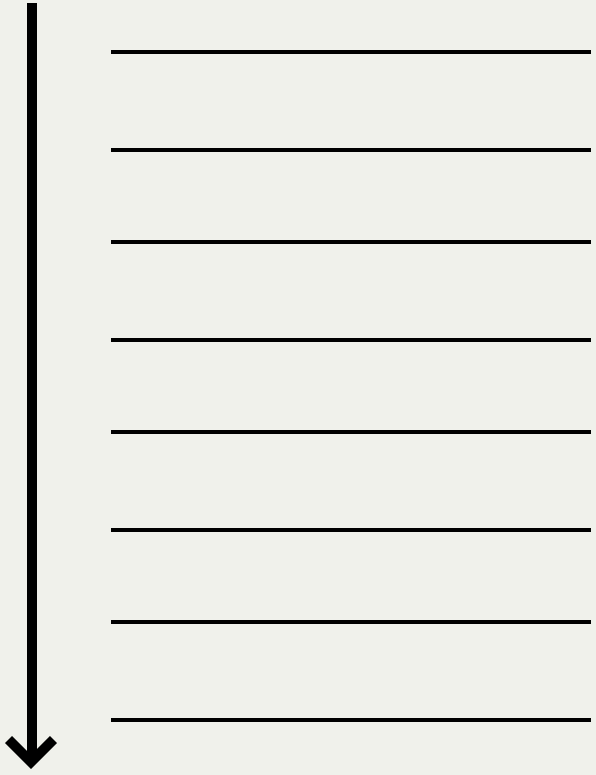
Exercise

Do the exercises found here

Conditionals

Program Execution

Typical Execution



JavaScript is an *asynchronous* programming language

It reads line by line from the top, and starts the execution of each line in order (but won't wait for each line to complete)

What are conditionals?

What are conditionals

- Conditional statements execute or skip parts of a program based on the value of an expression
- These are the decision points of your code, or the "branches"
- They rely quite heavily on boolean(ish) values

Why do we use them?

Why do we use them?

- To make decisions with code
- To allow "state" to change how our program runs
- To make part of our code optional

The *if* statement

The *if* statement

- This is the fundamental control statement that allows JavaScript to make decisions. It adds logic to the language
- It requires:
  - The *if* keyword
  - Parentheses with a condition in the middle
  - Curly brackets to identify what code should execute
- This is roughly how it runs

The *if* statement

```
if ( CONDITION GOES HERE ) {

  // CODE TO EXECUTE IF THE
  // CONDITION EVALUATES TO TRUE

}
```

The *if* statement

```
if ( 2 === 2 ) {
    console.log("That makes sense");
}

var favBook = "GEB";

if ( favBook === "GEB" ) {
    console.log("Good choice!");
}

var playerOneMove = "Rock";

if ( playerOneMove === "Scissors" ) {
    console.log("P1 played scissors");
}
```

The *if/else* statement

```
if ( CONDITION GOES HERE ) {

    // CODE TO EXECUTE IF THE
    // CONDITION EVALUATES TO TRUE

} else {

    // CODE TO EXECUTE IF THE
    // CONDITION EVALUATES TO FALSE

}
```

The *if/else* statement

```
if ( 2 === 2 ) {
    console.log("That makes sense");
} else {
    console.log("Uh oh");
}

var myNumber = 42;
if ( myNumber > 0 ) {
    console.log("The number is positive");
} else {
    console.log("The number is negative");
}
```

if/else if/else statement

```
if ( FIRST CONDITION ) {
    // CODE TO EXECUTE IF FIRST
    // CONDITION EVALUATES TO TRUE
} else if ( SECOND CONDITION ) {
    // CODE TO EXECUTE IF SECOND
    // CONDITION EVALUATES TO TRUE
} else {
    // CODE TO EXECUTE IF NEITHER
    // CONDITION EVALUATES TO TRUE
}
```

if/else if/else statement

```
var vehicle = "Car";

if ( vehicle === "Car" ) {
  console.log("You are driving a car");
} else if ( vehicle === "Motorcycle" ) {
  console.log("You are riding a motorbike");
} else {
  console.log("We aren't sure what you are driving");
}
```

if/else if/else statement

- You can have as many else ifs as you need
- JavaScript will run the code in only the ***first*** passing condition's curly brackets
  - It will skip the rest

Operators

What are they?

What are operators?

In JavaScript, we have *comparison* operators and *logical* operators

Comparison operators compare two values (e.g. equal, greater than)

Logical operators are used to make more complex conditionals - for example, deciding based on two different facts (e.g. this AND that, this OR that)

Comparison Operators

Comparison Operators

Operator	Meaning	Example
==	Loose equality	4 == "4"
===	Strict equality	4 === 4
!=	Loose inequality	1 != "5"
!==	Strict inequality	8 !== 2
>	Greater than	14 > 10
>=	Greater than or equal to	19 >= 14
<	Less than	10 < 242
<=	Less than or equal to	214 <= 344

== vs. ===

What is the difference between == and ===?

The answer is based on something called *type coercion*

When you use the threequals sign, it demands the values are the same AND the types are the same

When you use the double equals, it demands that the values can be the same (it will coerce the types for you)

== vs. ===

So, type coercion is when a data type is converted into a different type (e.g. 2 into "2")

I almost always use **===** so that I can be certain of my types, and I would rather be in charge of converting between types if that is necessary

You need to remember much less, too (e.g. Strict Equality vs. Loose Equality)

Logical Operators

Comparison Operators

Operator	Meaning	Example
&&	AND	3 === 3 && 2 === 2
||	OR	4 === 1 || 4 === 4
!	NOT	!false

&& (AND) Truth Table

CONDITION 1	CONDITION 2	RESULT
false	false	false
true	false	false
false	true	false
true	true	true

The && (AND) Logical Operator requires the conditions on BOTH sides to evaluate to true in order to pass

|| (OR) Truth Table

CONDITION 1	CONDITION 2	RESULT
false	false	false
true	false	true
false	true	true
true	true	true

The || (OR) Logical Operator requires the conditions on AT LEAST ONE side to evaluate to true in order to pass

! (NOT) Truth Table

VALUE	RESULT
false	true
true	false

The ! (NOT) Logical Operator takes one value and turns it into the opposite boolean representation

|| (OR) Logical Operator

```
var lang = "HTML";

if (lang === "HTML" || lang === "CSS" || lang === "JS") {
  console.log("You are talking about a frontend language");
} else {
  console.log("It is probably a backend language");
}
```

&& (AND) Logical Operator

```
var userNameExists = true;
var correctPassword = true;

if (userNameExists === true && correctPassword === true) {
  console.log("You are logged in");
} else {
  console.log("Something went wrong");
}
```

! (NOT) Logical Operator

```
var hasAccount = false;

if ( !hasAccount ) {
  console.log("You can create an account");
} else {
  console.log("You already have an account");
}
```

Exercise

Do the exercises found here

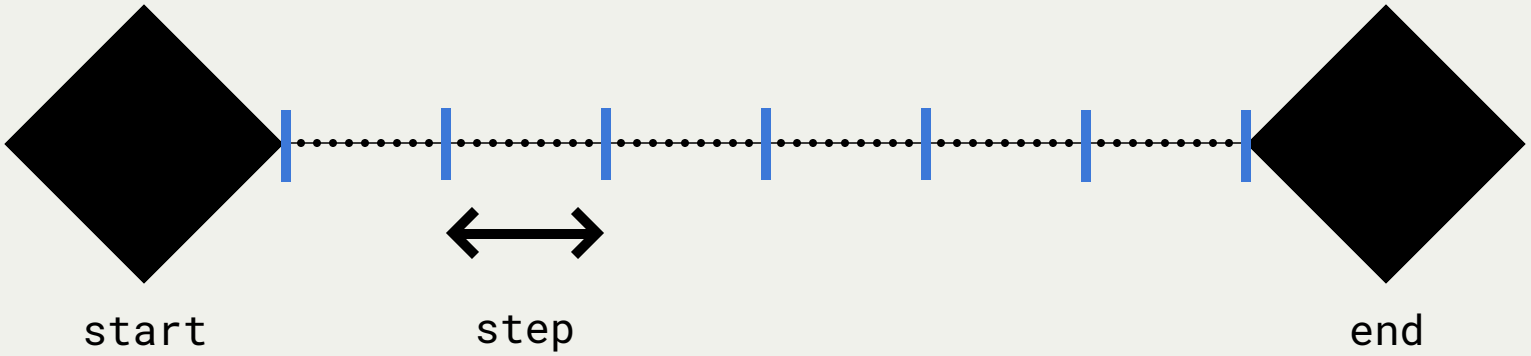
Loops

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

Homework

- Finish all exercises from class
  - Variables and Primitive Data Types
  - Conditionals
  - Loops
- Upload your homework to GitHub
- Prepare for next lesson

Homework (Extra)

- Read You Don't Know JS: Types & Grammar
- Read Eloquent JavaScript
- Read Speaking JavaScript

A quick note on homework

- There is way more than we expect you to complete!
- As long as you have a serious go, that is okay
- You'll get out what you put in!
- MDN is a great place to look for documentation

What's next?

- Collections in JavaScript
  - Arrays
  - Objects
- Functions
  - Parameters/arguments
  - Return values
- Scope in JavaScript

Questions?

Thanks!