Before we begin...

Open   →   https://bit.ly/jsd-class-01

Zoom   →   Videos On

Welcome!

Agenda

- Review and Homework Recap
- Variables
- Conditionals
- Loops
- Maybe:
  - Structural Data Types
  - Advanced Loops
  - Functions
- Exit Ticket

Review

Structural Data Types

What are structural data types?

- They are structures that JavaScript has to organise data
- They can contain other data types (both primitive and structural data types)
- They have distinguishable parts (e.g. a first item, a key etc.)

What structural types do we have?

- Objects
  - This includes: arrays, dates, maps, and sets plus a whole lot more (you can even create your own)
- Functions

Arrays, Objects and Functions will be the only ones we use today

Our First Type Structural Data Types

In JavaScript, when people think of structural data types, two main ones come to mind.

Arrays: Ordered data, that you access with an *index*

Objects: Unordered data that you access with a *key*

```javascript
const myArray = [ "A", "B", "C" ];

const myObject = {
  firstKey: "firstValue",
  secondKey: "secondValue"
};
```

Arrays

What are arrays?

- They are lists that can be filled with any data type
    - Both primitive and structural
- They are ordered and you access data with an index
    - An index is a zero-based number (meaning the first item is index 0, the second item is index 1 and so on)
- They are able to be iterated through (meaning loops through)

Creating Arrays

```javascript
const emptyArray = [];

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

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

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

Accessing Elements

```
const letters = [
  "A",
  "B",
  "C",
];

letters[0]; // "A"
letters[1]; // "B"
letters[2]; // "C"
letters[3]; // undefined
```

Reassigning Elements

```javascript
const instruments = [
  "The Great Stalacpipe Organ",
  "Stylophone",
  "Ondes Martenot",
  "Sharpischord",
  "Hydraulophone",
  "Pyrophone"
];

weirdInstruments[0] = "Roli Seaboard";
weirdInstruments[3] = "Makey Makey Banana Piano";
weirdInstruments[ weirdInstruments.length - 1 ] = "OP1";
```

Getting the length

```
const instruments = [
  "The Great Stalacpipe Organ",
  "Stylophone",
  "Ondes Martenot",
  "Sharpischord",
  "Hydraulophone",
  "Pyrophone"
];

instruments.length; // 6
instruments[ instruments.length - 1 ]; // "Pyrophone"
```

Looping through Arrays

```javascript
const ordinals = [
  "Zeroeth",
  "First",
  "Second",
  "Third"
];

ordinals[0]; // "Zeroeth"
ordinals[1]; // "First"
ordinals[2]; // "Second"
ordinals[3]; // "Third"

// Fair bit of consistency there
//   We could generate these indexes using a loop!
```

Looping through Arrays

```
const ordinals = [
  "Zeroeth",
  "First",
  "Second",
  "Third"
];
```

If we were to loop through this array, what would our:

- Starting number be?
- Ending condition be?
- Step be?

Looping through Arrays

```
const ordinals = [
  "Zeroeth",
  "First",
  "Second",
  "Third"
];

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

Looping through Arrays

```javascript
const ordinals = [
  "Zeroeth",
  "First",
  "Second",
  "Third",
  "Fourth"
];

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

Properties & Methods

```javascript
const ordinals = [
  "First",
  "Second"
  "Third"
];

ordinals.length; // 3

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

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

ordinals.indexOf("Second"); // 1
```

Do the exercises <u>here</u>, please!

See you in 10 minutes!

Objects

What are objects?

- Objects are a collection of key-value pairs (like a word and a definition in a dictionary)
  - They are unordered!
- They, like arrays, can store any combination of data types

Why use objects?

- They allow us to effectively interact with large amounts of data
- They allow for encapsulation and modularity:
  - It is a way to group functionality and data
  - It makes sharing your code a lot easier

Creating Objects

```
const emptyObject = {};

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

Creating Objects

```javascript
const bookSeries = {
  name: "In Search of Lost Time",
  author: "Marcel Proust",
  rating: 5,
  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

```
const bookSeries = {
  name: "In Search of Lost Time",
  author: "Marcel Proust",
  rating: 5,
  books: [
    "Swann's Way",
  ]
};

const name = bookSeries.name; // "In Search of Lost Time"
const author = bookSeries.author; // "Marcel Proust"
const rating = bookSeries.rating; // 5

const firstBook = bookSeries.books[0]; // "Swann's Way"
```

Changing Values

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

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

Adding new Values

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

movie.language = "Hungarian";
movie.ratingOutOfFive = 10;
movie.parts = 12;
```

Nested Objects

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

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

Complex Data Structures

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

for ( const i = 0; i < marxFamily.length; i += 1 ) {
  const brother = marxFamily[i];
  console.log(brother.name, brother.birthYear);
}
```

Do the exercises <u>here</u>, please!

See you in 10 minutes!

Functions

What are functions?

The main building blocks of programs - they are reusable sections of code (almost like subprograms).

They are very useful for when we need to perform a single action in many places of the script, and for reducing repetition.

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.

Functions are also the way we execute code based on events (e.g. clicking, scrolling, time etc.)

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?

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

Declaring Functions

```javascript
// Function Declaration

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

function add() {
  console.log(2 + 2);
}

function subtract() {
  console.log(10 - 3);
}
```

Declaring Functions

```javascript
// You will also see these two variations!

// Function Expression
const speak = function() {
  console.log("Hello");
}

// Arrow Function
const add = () => {
  console.log(2 + 2);
};
```

Calling Functions

```javascript
// Function Declaration

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

// Function Call Site

sayHello();
```

Parameters and Arguments

Our functions aren't dynamic just yet. This is where we need to see *parameters* and *arguments*.

They are how we provide a function with extra data or information.

We listen for *parameters* in the function declaration.

We provide *arguments* at the call site.

Parameters and Arguments

```javascript
// `name` is our parameter

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

// "Jacques" is our argument

sayHello("Jacques");
```

Parameters and Arguments

```javascript
function add(numOne, numTwo) {
  const solution = numOne + numTwo;
  console.log(solution);
}

add(5, 10); // numOne will be set to 5, numTwo will be set to 10

add(3, 18); // numOne will be set to 3, numTwo will be set to 18
```

`return` Values

Sometimes your function calculates something and you want the result!

`return` values allow us to do that.

We can store the results of calculation through the use of `return` values (think of `.toUpperCase();`);

return Values

```javascript
function squareNumber(num) {
  const square = num * num;
  return square;
}

const squareOfFour = squareNumber(4);

console.log(squareOfFour); // 16
```

return Values

```
function squareNumber(num) {
  const square = num * num;
  return square;
}

const squareOfFour = squareNumber(4);
const squareOfTwelve = squareNumber(12);

console.log(squareOfFour + squareOfTwelve); // 160

console.log(
  squareNumber(8) + squareNumber(11)
); // 185
```

return Values

```javascript
function cube(num) {
  return num * num * num;
}

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

const result = double(cube(4)); // 128
```

return Values

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

const userTwo = {
  admin: false
};

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

isAdmin(userOne); // true

isAdmin(userTwo); // false
```

Passing in Variables

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

const firstNumber = 10;
const secondNumber = 24;

addTwoNumbers(firstNumber, secondNumber); // 34
```

Function Guidelines

Follow F.I.R.S.T principles:

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

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

Do the exercises <u>here</u>, please!

See you in 10 minutes!

That's all!

Homework

- Finish off in-class exercises
- Arrays
- Objects
- Functions
- Extra: Begin reviewing the next lesson's content

What's next?

- Pseudocode
- Advanced Functions
    - Callbacks
    - Scope and Hoisting
    - Closures
    - Higher Order Functions
    - Rest Parameters
    - Spread Operator

Thank you!