**JS From zero to hero**

**Datatypes**

There are different types of **values**

Firstly: **Objects**

secondly: **primitive datatypes** (everything else)

there are 7 types of **primitive datatypes**:

**1 number**these are ALWAYS floating point numbers

**2 string**sequence of characters. Always put them in paranthesese

**3 boolean**

True or false.

**4 undefined**an empty value. A value that has not yet been assigned a value.

**5 Null**also means empty value but a bit different

**6 symbol**Symobl. Value that is unique and cannot be changed.

**7 BigInt**Very large integers. Larger than number could hold.

Javascript has dynamic typing. This means that The Value has Type, not the variable. This also means that any variable we store can have its type changed later in the porgram.

**Let, const and var**

Let and const are modern javascript and var is old.

**Let** are mutable. (let age. Because age can change)

**Const** is immutable (const birthyear for example cannot change)  
the course and general proper practice is that any value should be initiallized as a const and **only** when you it turns out you want it to be mutable, than turn it into a **let.**

**Var.** This is the old way of creating variables and is a lot like **let. Let** is block scoped and **var** is function-scoped. **We should never use** var though.

**Basic operators**

Basic stuff. You know this

**Operator precedence**

Google mdn operator precedence.

Hoe hoger het getal hoe hoger de precedence. Let ook op de middelste column die aangeeft of code van links-rechts gaat of rechts-links gaat.

**String and Template literals**

**Type Conversion and Coercion**

Conversion = when we manually converse types.

Coercion is when javascript does it (accidentily) behind the curtains

You can convert numbers, strings and booleans to each other. Converting anthing to undefined or null doesnt make sense.

**Truthy and Falsy statements**

Or conversion with booleans,  
5 falsy values. Values that are not exactly false but turn false when we convert them to boolean.

The 5 falsy values are: 0, ‘’ (empty string), undefined, null, NaN

These are not false but become false when converted to boolean.

Any positive number or filled string becomes true when converted to a boolean

**Equality Operators == vs ===**

=== strict equality operator. Only returns true when both sides are EXACTLY the same.

== loose equality operator. It will use typecoercion to make things similar.

This is confusing for many people. Especially the loose operator results in many strange bugs.

**Boolean logic**

And, Or and NOT operators

**The Switch Statement**

Switch vs else if basically

**Statements and expressions**

An expression is anything that produces a value.

**The Conditional**

**Functions overview**

3 types of functions

1. function declaration – a function that can be used before its declared.

*function* percentageOfWorld1(*populationMillions*) {

return (populationMillions / 7900 \* 100);

}

1. Function expression – Essentially a function *value* stored in a variable

*const* percentageOfWorld2 = *function* (*populationMillions*) {

    return (populationMillions / 7900 \* 100);

}

1. Arrow function. Great for a quick one-line function.

*const* percentageOfWorld3 = *populationMillions* =>

   populationMillions / 7900 \* 100;

**92 Scoping and scope in javascript**  
  
Scope  
How our progams variables are organized and accessed

Lexical scoping  
Scoping is controlled by placement of function and blocks in code

Scope  
Space or environment in which a certain variable is declared. There is **global** scope, function scope and block scope.

scope of a variable  
Region of our code where a certain variable can be declared.

99 primitives vs objects   
AKA  
storing in the call stack vs storing in the heap

A screenshot of a computer

Description automatically generated

Section 9

103 destructuring arrays

Destructuring arrays is unpacking an arrays in different variabled.

!!!  
remember declaring a variable creates the variabled. It declares to exist  
 initalizing a variable, is assigning a value to a variable (for the first time)

So: first you *declare*, than you initalize (or you do both at the same time)

Arrays : [ ]

Objects: { }

104 destructuring object:

Pretty coplex. Or badly explained

**105 Spread operator**

Basically if youra rray is named Array1.

And you type ...Array1 it will unpack the entire array.

This spread operator ... works on ANY iterables:

Arrays, strings, maps, sets etc

It really helps when you dont know your array length!

107 && and shortcircuiting

|| OR will return the first truthy value

If everything is falsy it will return the last value (which is falsy)  
**Practically this is used to set default values**

&& If everything is truthy, it will return the last value.

If anything is falsy it return this falsy value.  
Practically, you can execute some code when the first operand is true.

108 Nullish coalescent operator

Is literally the same as the || or operator except is works with Nullish values rather than Falsy values  
Falsy values: Null + undefined + 0 + ‘ ‘ (empty string)

Nullish values: Null + undefined (not 0 and ‘ ‘ (empty string)

// 109 Logical Assignment operators

Very new and modern 3 new operators

restaurant.numGuests = 23;

const guests1 = restaurant.numGuests ? restaurant.numGuests :10;

console.log(guests1);

==

const guests2 = restaurant.numGuests||10;

console.log(guests2);

==

rest1.numGuests ||= 10;

//FIRSTLY based of  || -> the OR assignment operator

// Using the || operator to set a default

rest2.numGuests = rest2.numGuests || 10

rest1.numGuests = rest1.numGuests || 10

// This is literally the same but even Shorter!

rest1.numGuests ||= 10;

rest2.numGuests ||= 10;

console.log(rest1);

console.log(rest2);

//SECONDLY variant of ?? -> Nullish assignment operator

rest1.numGuests ??= 10;

rest2.numGuests ??= 10;

console.log(rest1);

console.log(rest2);

// Thirdly:variant of && -> the AND assignment operators

// If the restaurant has an owner, we want to replace it with the string anonymous

rest1.owner = rest1.owner && '<ANONYMOUS>';     // returns undefined, because rest1.owner is undefined

rest2.owner = rest2.owner && '<ANONYMOUS>';     // returns anonymous, because rest2.owner is truthy

rest1.owner &&= '<ANONYMOUS>'

rest2.owner &&= '<ANONYMOUS>'

113 optional chaining

Restaurant.mon?.open?

De? Is basically een optional check.

De ? checkt of het element links bestaat. Als dat niet zo is, dan wordt het gedeelte rechts ook niet gelezen/uitgevoerd

De optional chain wordt bijna altijd gebruikt in combinatie met de nullish asignment operator. Like zo:

Console.log(users[o]?.name?) ?? “Array is empty”

Als users array een 1e entree heeft en die heeft een naam. Dan is alles ok.   
Maar als een van beide niet bestaat, dan krijg je bericht: “Array ios empty” terug. Dit is super handig en nuttig.

114 Looping Objects, keys, values and entries

An entrie exist of a key/value pair. It can be used to loop over objects.

Object.keys(days) // day is a list of days the restaurant is open

Returns

116 SETS

A set is a collection of unique values (no duplicates). You can add duplicates and it will work, but it won’t actually show the same value twice.

This work with any **iterable**. So making a set of a word just pulls the word apart in different letters.

Conclusion at the end. Sets are very similar to arrays, but they are handy if you need to have unique values. Otherwise, just keep using arrays.

117 MAPS

Maps are a lot more useful than sets. Joans said: maps are very powerful. They store information in key-value pairs, like objects, however contrary to objects map can store **keys** of any type even objects (whereas objects virtually always have string values.)

119 **Which data structure should to use**

**Where can data come from:**

* from the program itself. Data written in the source code, status messages
* From the UI. Data input from the user or information from the DOM
* External sources – a web api
* This often results in **COLLECTIONS OF DATA**
* **DATA STRUCTURES** is where we store these collections.

**Simple list? 🡪 Array or sets**

**Key/value pairs 🡪 Objects or Maps (keys allow us to describe values**

Firstly the simple structures of values

**Arrays** use when you need ordered list of values (might contain duplicates

Use when you need to manipulate data

**Sets** Use when you need to work with unique values

Use when high performance is important

Use to remove dupicates from arrays

Sets are not meant to replace arrays, instead they are meant to complement each other.

Next objects and maps, when we need to describe values using keys

**Objects METHODS** use when you need **METHODS**

**JSON** use when you need to use json

more traditional key/value store (“abused” objects)  
Easy to write and access values with . and []

**Maps** Use when you simply need to map keys to values

Use when you need keys that are not strings

Better performance  
Keys can have **any datatype**Easy to iterate   
Easy to compute size

A screenshot of a computer

Description automatically generated