POWERING POTENTIAL

Types

→ JavaScript Fundamentals

INTRODUCTION

In this module, you will learn to:

- Declare variables
- Understand types
  - Primitive types
  - Strings
  - Numbers
  - Booleans
  - Undefined
  - Nulls
  - Symbol
  - Reference types

Declaring variables

- Declaring variables
  - **`const`**, **`let`** and **`var`**
  - With and without assignment
  - Do not use implicit declaration

Variable Declarations

```
x = 10;       // implicit - DO NOT USE
let y;        // explicit without assigment
let y = 15;   // explicit with assignment
const z = 10; // constant with assignment
```

- **`let`** – a block-scoped variable (don't worry – we'll discuss what block-scoped means later)
- **`const`** - the same as **`let`**, but must be initialised at declaration and cannot be changed
- **`var`** – a function-scoped variable whose declaration is hoisted and can lead to confusing code! To be avoided now that we have **`let`** and **`const`**
- What should you use? **`const`** where possible. **`let`** when you need it to change

Declaring variables

- Variable names
  - Start with a letter , "_" or "$”
  - May also include digits
  - Are case sensitive
  - Cannot use reserved keywords
  - E.g. int, else, case
- Best practice is to use camelCase for variable names

JavaScript types

- Dynamically typed
  - Data types not declared and not known until runtime
  - Variable types can mutate
- Interpreted
  - Stored as text
  - Interpreted into machine instructions and stored in memory as the program runs

- Primitive data types
  - Boolean
  - Number
  - String
  - Undefined
  - Null
  - Symbol
- Object

Primitives and Object types

- JavaScript can hold two types:
- **Primitives**
  - Primitive values are immutable pieces of data
  - Their value is stored in the location the variable accesses
  - They have a fixed length
  - Quick to look up
- **Object**
  - Objects are collections of properties
  - The value stored in the variable is a reference to the object in memory
  - Objects are mutable

The typeof operator

- The **`typeof`** operator takes on parameter the value to check

The typeof operator

```
const TYPE_TEST = "string value";
alert(typeof  TYPE_TEST)     //outputs “string”
alert(typeof 95)          //outputs “number”
```

- Calling **`typeof`** on a variable or value returns one of the following:
  - number
  - boolean
  - string
  - undefined
  - symbol
  - object (if a null or a reference type)

The undefined type

- A variable that has been declared but not initialised is **`undefined`**

The undefined type

```
let age;
console.log(typeof age);  //returns undefined
```

- A variable that has not been declared will also be **`undefined`**
  - The **`typeof`** operator does not distinguish between the two

The undefined type

```
let boom;
console.log(typeof boom);  //returns undefined
```

null is not undefined

- **`null`** and **`undefined`** are different concepts in JavaScript
  - **`undefined`** variables have never been initialised
  - **`null`** is an explicit keyword that tells the runtime it is 'empty'

```
let userID = null;
console.log(userID); //returns null
```

- There is a foobar to be aware of with **`null`**:
  - **`undefined`** is the value of an uninitialised variable
  - **`null`** is a value we can assign to represent objects that don't exist

```
let userID = null;
console.log(userID == undefined); //returns true
```

The Boolean type

- Boolean can hold two values – **`true`** and **`false`**
- These are reserved words in the language:

```
let loggedOn = false;
console.log(loggedOn); //returns false
```

- When evaluated against numbers, you can run into issues
  - **`false`** is evaluated as **0**
  - **`true`** can be evaluated to **1**

The Number type

- Always stored as 64-bit values
- If bitwise operations are performed, the 64-bit value is rounded to a 32-bit value first
- There are a number of special values

Constant	Definition
Number.NaN or **NaN**	Not a number
Number.Infinity or **Infinity**	Greatest possible value (but no numeric value)
Number.POSITIVE_INFINITY	Positive infinity
Number.NEGATIVE_INFINITY	Negative infinity
Number.MAX_VALUE	Largest possible number represented in the 64-bits
Number.MIN_VALUE	Smallest possible number represented in the 64-bits

The String type

- Immutable series of zero or more unicode characters
  - Modification produces a new string
  - Can use single (') or double quotes (") or backticks (`)
  - Primitive and not a reference type
- String concatenation is expensive
- Back-slash (\) used for escaping special characters
- As a rule, always use backticks (`)

Escape	Output
\'	'
\"	"
****	\
\b	Backspace
\t	Tab
\n	Newline
\r	Carriage return
\f	Form feed
\ddd	Octal sequence
\xdd	2-digit hex sequence
\udddd	Unicode sequence (4-hex digits)

String Concatenation and Interpolation

- Adding two (or more strings) is an expensive operation due to the memory manipulation required
- To concatenate a string the + operator is used

```
let str1 = "5 + 3 = ";
let value = 5 + 3;
let str2 = str1 + value
console.log(str2); // 5 + 3 = 8
```

- Template literals (introduced in ES2015) allow for strings to be declared with JavaScript expressions that are evaluated immediately using ${} notation

```
let str2 = `5 + 3 = ${5 + 3}`;
console.log(str2); // 5 + 3 = 8
```

String functions

- The String type has string manipulation methods, including:

Method	Description
indexOf()	Returns the first occurrence of a character in a string
charAt()	Returns the character at the specified index
toUpperCase()	Converts a string to uppercase letters

- Method is called against the string variable

```
let str = "Hello world, welcome to the universe.";
let n = str.indexOf("welcome");
```

QuickLab Chapter 13

- Exploring types
- Create variables of a number type
- Using methods of the number object
- Creating variables of a string type
- Using string functions to manipulate string values

REVIEW

- **Primitive variables**
  - Value types
- **Understand types**
  - There are six primitive types and object
- **Types can mutate**