DATA TYPES, SELECTION AND ITERATION

Lesson Objectives

- Declare and use variables and data types in JavaScript
- Understand JavaScript datatypes
  - Understand arithmetic, relational and logical operators.
  - Understand implicit and explicit type conversions.
  - Learn to use Math object.
- Use JS selection logic and syntax
- Learn to use JavaScript loop syntaxes

Assigning values to a variable

- When variables are declared, their default value is 'undefined’

```
let total;
console.log(total) //
undefined

total = 5;
console.log(total) // 5
total = total + 10;

console.log(total); // 15
```

Declaring & assigning constants

- To declare a constant (unchanging) variable, use `const` instead of `let`

```
const WEEK_DAYS = 7;
const PI = 22/7;
```

- Convention is upper case for constants known in advance
  - can be used throughout program
  - easy to update in single place
  - Lower case for (local) const variables that you do not expect the program to change (reassign)

  const name = friendList[0];

Demo

- Write code to output 5+5
- Update code to save some integer values on **constants** X and Y and print the sum to the console (console.log).
- Update code to save some integer values on re-assignable **variables** x and y and print the sum.

Exercise

- We will do the environment setup together
- Download and install VSCode
- Download and install NodeJS
  - Try to run some JavaScript code on Node.js REPL

Main point: JavaScript Variables

- JS variables should be declared using the keywords const or let
- JavaScript variables are loosely typed

Main point preview: JavaScript Data Types

- JS has 8 data types, the most important are:
  - Number
  - String
  - Boolean
  - Object (includes Array)
  - null and undefined
- Math and relational operators are like Java

Data Types

- All programing languages have data types
  - Usage can be strict or loose
- In JavaScript, a variable gets its type based on current value
- 6 primitive data types: **`string`**, **`number`**, `bigint`, **`boolean`**, **`undefined`**, and `symbol`.
  - also **`null`**, which is seemingly primitive, but is a special case for every **`Object`**.
- can put any type in a variable.

```
// no error
let message = "hello";
message = 123456;
```

- Programming languages like this are called *“dynamically typed”*
  - *Also called “loosely typed” or “weakly typed”*

Number

- The *number* type represents both integer and floating-point numbers.
- Besides regular numbers, there are so-called “special numeric values” which also belong to this data type: `Infinity`, `-Infinity` and `NaN`.

```
let n = 123;
n = 12.345;
console.log( 1 / 0 ); // Infinity
console.log( "not a number" / 2 ); // NaN
```

- `BigInt` type was added to the language to represent integers of arbitrary length. (reading)

String

- A string in JavaScript must be surrounded by quotes.

```
let str = "Hello";
let str2 = 'Single quotes are ok too (but we prefer double)’;
let phrase = `can embed another ${str} in a backtick quote`;
```

- There is no `char` type
- A string may consist of zero characters (be empty), one character or many of them.
- Double or single quotes allowed
  - We will favor double quotes

Boolean (logical type)

- The `boolean` type has only two values: `true` and `false`.

```
let isLazy = false;
let isHealthy = true;
```

null and undefined

- special values `null` and `undefined` are special types as well.
- `null` is not a “reference to a non-existing object” or a “null pointer” like in Java
  - special value which represents “nothing”, “empty” or “value unknown”.
- The meaning of `undefined` is “value is not assigned”.
  - If a variable is declared, but not assigned, then its value is `undefined`.
- Programmers assign null; the compiler assigns undefined

```
let name = null;
let age;
console.log(name,age) // null, undefined
```

Objects and Symbols

- Object is a complex data type.
  - Objects are used to store collections of data and more complex entities

- All other types are called “primitive” because their values can contain only a single thing (be it a string or a number or whatever)

Interactions: alert, prompt, confirm

- In browser environment, there are built-in **functions** to interact with the user `alert`, `prompt` and `confirm`.
- In Node.js environment we will be using external `prompt-sync` module for input from the console.

```
const prompt = require("prompt-sync")();

let name = prompt("What is your name?: ")
console.log("Hi ", name);
```

Exercise

- create a file, prompt.js
- add the following code and run the file
  - First need to install this function in node for the prompt
  - npm install prompt-sync
  - If do in parent directory do not need to do it inside any subfolders

```
const prompt = require("prompt-sync")();

let name = prompt("What is your name?: ")
console.log("Hi ", name);
```

Type Conversions

- Most of the time, operators and functions automatically convert the values given to them to the right type.
  - For example, `alert` and console.log automatically convert any value to a string to show it.
  - Mathematical operations convert values to numbers.

- There are also cases when we need to explicitly convert a value to the expected type.
  - E.g., when get user input or data over a network

String Conversion

- String conversion happens when we need the string form of a value.
  - For example, `console.log(value)` does it to show the value.
- We can also call the String(value) function to convert a value to a string:

```
let b = true;
let n = 5;

console.log(typeof b, typeof n) // boolean number

let s1 = String(b);
let s2 = String(n);

console.log(typeof s1, typeof s2) // string string
```

Numeric Conversion

- Numeric conversion happens in mathematical functions and expressions automatically.
  - For example, when division / is applied to non-numbers:

```
alert( "6" / "2" ); // 3, strings are converted to numbers
```

  - We can use the Number(value) function to explicitly convert a value to a number:

```
let str = "123.33";
let num = Number(str); // becomes a number 123.33
num = parseFloat(str); // becomes a number 123.33
num = parseInt(str); // becomes a number 123
```

  - Favor Number unless have a specific need for parseInt/parseFloat

Numeric conversion rules

Value	Becomes…
undefined	NaN
null	0
true and false	1 and 0
string	Whitespaces from the start and end are removed. If the remaining string is empty, the result is 0. Otherwise, the number is “read” from the string. An error gives NaN.

```
console.log( Number("   123   ") ); // 123
console.log( Number("123z") );      // NaN (error reading a number at "z")


console.log( Number(true) );        // 1
console.log( Number(false) );       // 0
```

User input may need numeric conversion

- User input always comes as string, even when user may have entered a number.
- Before arithmetic operation can be performed, string should be converted to numeric type.
  - `Number(string)`
  - `parseInt(string)`
  - `parseFloat(string)`
  - *`Using unary (+) operator (reading)`*

- See example: demos/d11iteration/parsing_user_input.js

Boolean Conversion

- Boolean conversion is the simplest one.
  - It happens in logical and relational operations (covered later)
  - But can also be performed explicitly with a call to Boolean(value).
- The conversion rule:
  - Values that are intuitively “empty”, like 0, an empty string, null, undefined, and NaN, become false.
  - Other values become true.

```
console.log( Boolean(1) ); // true
console.log( Boolean(0) ); // false

console.log( Boolean("hello") ); // true
console.log( Boolean("") ); // false
```

Operations & Operators

- Different set of operations can be performed on a variable based on its data type
  - Arithmetic operations use operators (+, -, *,**, /, %, ++, --)
    - *When at least one of the operand type is string, + operator will perform string concatenation.*
  - Comparisons use relational operators (===, !==, ==, !=, >, <, >=, <=)
  - Logical operations use operators (&&, ||, !)

Arithmetic Operations

- These are similar operations as in mathematics (algebra)
  - same precedence
- modulus operator (%) returns remainder
- division (/) operator returns quotient.
- Multiplication (*) operator must be explicitly used
- exponentiation operator a ** b multiplies a by itself b times.
- What is result of following arithmetic operations?
  - 2-9+8-6+5
  - 2-9+8-6*5
  - parens always a good idea to remove any ambiguity

More assignment operators

- These are merely shortcuts

Operator	Example	Equivalent
+=	x += 2	x = x+2

- Same rule for -, *, / and % operators

- x = x + 2:
- x +=2

Increment and Decrement Operators

- Shortcuts to increment and decrement current value by 1

  ++ count; => count += 1; => count = count + 1;

  -- count; => count -= 1; => count = count – 1;

- Pre vs Post, increment and decrement

  ++ count vs count ++

  -- count vs count –

  - Avoid using these in expressions (and in general)

Operator Precedence Revisited

Operator(s)	Name(s)
`()`	parentheses
`- (unary)`	negation
`++  --`	increment, decrement
`*  /  %`	multiplication, division, modulus
`+  -`	additions, subtraction
`=  +=  -=  *=  /=  %=`	assignments

Math Object

- Built-in object part of JavaScript language
  - functionality for mathematical computations

- See example: demos/d11iteration/math.js

Exercise

- Following program asks user to input temperature value in degree Celsius and outputs the result in degree Fahrenheit. Make this program run on your machine.

```
const prompt = require("prompt-sync")();
const tempInCelsius = prompt("Enter value in celsius: ")
const tempInFahrenheit = 9/5*parseFloat(tempInCelsius)+32;
console.log(tempInFahrenheit);
```

- Write a program that computes volume of a cylinder based on user inputs for radius and height of a cylinder, using formula $v = \pi r^2 h$

Relational operators

- As the name implies, these operators compare two values and return either true or false (Boolean values).
- <, >,<=, >=, ==, ===, !=, !==

- Strings in JavaScript are compared based on so-called “dictionary” or “lexicographical” order.

- See example: demos/d11iteration/relational_operators.js

Comparing different types

- When comparing values of different types, JavaScript converts the values to numbers.

```
console.log( '2' > 1 ); // true, string '2' becomes a number 2
console.log( '01' == 1 ); // true, string '01' becomes a number 1
```

- A strict equality operator === checks the equality without type conversion.
  - In other words, if a and b are of different types, then a === b immediately returns false without an attempt to convert them.
  - Always use === instead of ==

- Generally, comparison of different types is a mistake or poor design

Logical Operators

- Logical operators (&&, ||, !), usually used with Boolean (logical) values.
  - Although they are called “logical”, they can be applied to values of any type, not only boolean.
  - *Their result can also be of any type (see slide on short-circuit evaluation).*

- Recall, relational operators return Boolean values.
  - logical operators can be to used to combine two or more relational expressions.

- See example: demos/d11iteration /logical_operators.js

Short-circuit evaluation

- The && and || operators return the value of one of the operands, so if used with non-Booleans, they will return non-Boolean value.
  - See example: demos/d11iteration/short_circuit_or.js
  - Avoid doing this practice (unless highly experienced, and even then)

Truthy & Falsy

- In JavaScript, any value can be used as a Boolean
  - "**falsy**" values: 0, 0.0, NaN, "", null, and undefined
  - "**truthy**" values: anything else

Main point: JavaScript Data Types

- JS has 8 data types, the most important are:
  - Number
  - String
  - Boolean
  - Object (includes Array)
  - null and undefined
- Math and relational operators are like Java

Main point

- Data types determine the operations that can be carried out on data. The same operator can do different operations depending on the data type. *Science of consciousness: We can compare a human nervous system to a data type and TM to an operation on the nervous system. TM operates in same way for every human nervous system.*

Control logic

- Sequence
- **Selection**
- Repetition

Selection logic

- Selective execution can be done with the use of **if**(...) statement.
- **if** statement may contain an optional “**else**” block. It executes when the condition is false.
- The ***if...else if...else*** a special statement that is used to combine multiple *if...else* statements.
- **nested-if:** A nested if is an if statement that is the target of another if or else. Nested if statements means an if statement inside an if statement.

The “if” statement

- Selective execution can be done with the use of `if(...)` statement.
  - The `if(...)` statement evaluates a condition in parentheses and converts the result to a boolean

```
if(condition){
// statements;
}
```

recommended to wrap your code with curly braces {…}, even if there is only one statement to execute.

The “else” clause

- `if` statement may contain an optional “`else`” block. It executes when the condition is false.

```
if(condition){
// statements;
}else{
// different statements;
}
```

Several conditions: “else if”

- Sometimes, we’d like to test several variants of a condition.
  - The `else if` clause lets us do that.

```
let year = prompt('In which year was ES6 released?', '');

if (year < 2015) {
  alert( 'Too early...' );
} else if (year > 2015) {
  alert( 'Too late' );
} else {
  alert( 'Exactly!' );
}
```

  - There can be many `else if` blocks. The final `else` is optional.
- *See example: demos\d11iteration\selection\selection_statements_else_if.js*

Nested If Statements

- There are times when we need to check for more conditions when prior conditions are met.
- One way to achieve this is using nested if statements.

```
let weather = prompt("Please enter weather outside");
let temp = prompt("Please enter current temperature");

if (weather == 'sunny') {
    if (temp < 80) {
        console.log("Good day for outdoor running")
    } else {
        console.log("Better use tread mill at home.")
    }
}
```

- Best to avoid more than one level of nesting if possible.

Scope of variables

- The scope of a variable determines how long and where a variable can be used.
- When const or let keywords are used, scope is within the block

```
let x = 5;
console.log(x);
if(x==5){
        let y = 2*x;
        console.log(y);
        console.log(x); // x is accessible here.
}
console.log(x);
console.log(y); // y is not accessible here.
```

- Declare y using `var` keyword in above code and see the change in output.

Exercise

- Write a program that accepts user age as input and output following based on the input
  - If age <= 0
    - print "please enter valid age"
  - if age is between 0 and 14
    - print "You can't drive yet."
  - if age is between 15 and 18
    - print "You can drive under supervision."
  - if age is 19 or higher
    - print "You can drive."

Main point

In programming, we encounter situations where we must make decisions based on conditions that determine the flow of code execution. We make use of relational and logical operators to make such decisions. Our programs produce expected results only when our decision logic is correct. *Science of Consciousness, as in programing, taking the right decision at the right point of time is also crucial to success in life. Our actions and decisions are spontaneously in the right direction when we have a good connection to the field of pure intelligence.*

Control logic

- Sequence
- Selection
- **Repetition**

Repetition/ Looping

Loops are used to execute the same block of code again and again, as long as a certain condition is met.

- **while** — loops through a block of code as long as the condition specified evaluates to true.
- **do…while** — loops through a block of code once; then the condition is evaluated. If the condition is true, the statement is repeated as long as the specified condition is true.
- **for** — loops through a block of code until the counter reaches a specified number.
- **for…in** — loops through the properties of an object.
- **for…of** — loops over iterable objects such as arrays, strings, etc.

Nested Loops – Loop inside of a loop.

while loop

- pre-test loop
- Can be used as both counter-controlled loop or sentinel-controlled loop.

```
while (condition) {
        statements;
}
```

- Counter-controlled
  - see example: *demos\d11iteration\loops\while_counter_controlled.js*
- Sentinel-controlled
  - see example: demos\d11iteration\loops\while_sentinel_controlled.js

  Careful! if you forget to update a loop counter or because of logic error loop counter is never reaching the exit condition, loop will be infinite.
- Ctrl-C Ctrl-C to break out of NodeJS process

do while

- post-test loop
- Better suited for sentinel-controlled loop
  - The statements inside do block is guaranteed to be executed at least once
- Example – updated sentinel controlled while loop.
  - See *demos\d11iteration\loops*\do_while.js
  - Notice: this version makes prompt at only one place, that is inside the do part of the do-while loop.

for

- Pre-test loop
- Better suited for counter-controlled loop
  - Updating counter is part of the syntax, hence you won't forget!

```
for (initialization; condition; update) {
      statements;
}
```

- See *demos\d11iteration\loops\for_loop.js*

Nested Loops

- Loop inside of a loop

```
for(let i = 1; i<=10; i++){
    let row="";
    for(let j=1; j<=10; j++){
        row += i*j +"\t";
    }
    console.log(row);
}
```

Exercise

- Write code to print following pattern on the console

```
11111
22222
33333
44444
55555
```

Main Point

Looping/ repetition is the control structure that powers computation. The same operations repeated over and over with slight changes to state information produces all the interesting effects of computation. *Science of consciousness: Self referral awareness, pure consciousness or the unified field aware of itself by its own nature is the source of all energy and matter. "Curving back on myself I create again and again."*