Module 1 - Intro to JS & Computer Programming

JavaScript as an Interpreted Language

- run code written in web browser → interpreter = JS Engine built in browser
- node.js: interpreter that is installed independently of browsers as an environment in operating system (macOS, Windows, or Linux)
- most JS engines use *Just In Time Compilation* technique (*JIT Compilation)*, which compiles code fragment during execution of program & allows increased performance (change virtually unnoticeable)

Client-Side vs Server-Side Programming

- Client-Side: code to be executed is loaded together w/ page in browser, on user's side, & interpreter, should be embedded inside an HTML document
- Server-Side: back-end, executed on servers, processing data (from databases), which after processing will be available on client side

Disadvantages

- limited in functionality for certain applications
- since code isn't compiled, it goes into browser in the same form to what we wrote ourselves → everyone can see solution in easy-to-read form & use it w/out our permission

Advantages

- very active/supportive community → easy to find solutions to common problems → tools that work w/ JS actively developed
- ready-to-use frameworks & libraries that provide most of commonly required functionalities + features
- JS doesn't require you to buy expensive tools
- *Dynamic Typing* Characteristic: we can store data of any type in a variable
- *Static Typing*: variable can only contain one type of variable during program execution

Online environment - sites that act as simple editor & runtime environment

Local Development Environment

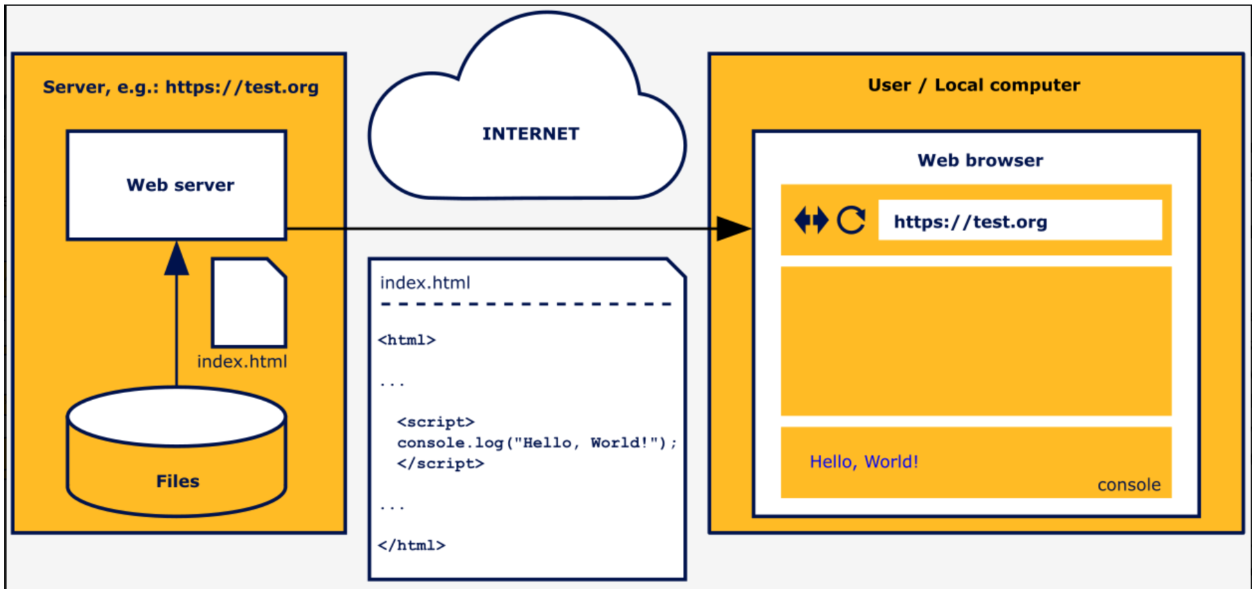
- Package Managers: enable management of libraries/components of development environment (npm, yarn)
- Task Runners & Module Bundlers: automate process of software development & merge resulting doe from many files & libraries (Grunt, Webpack)
- Testing Framework: allows for automatic testing of correctness of our program in search of potential errors (Mocha, Jasmine, or Jest)
- Security Analyzers: used to control security of our solution (Snyk, RetireJS, OWASP Dependency Check)

- Minimal Tools Needed
    - Code Editor: application that writes plan text
    - Interpreter: runtime environment from program
    - Debugger: tool that allows you to slow down execution of program

console.log("Hello World")
HTML

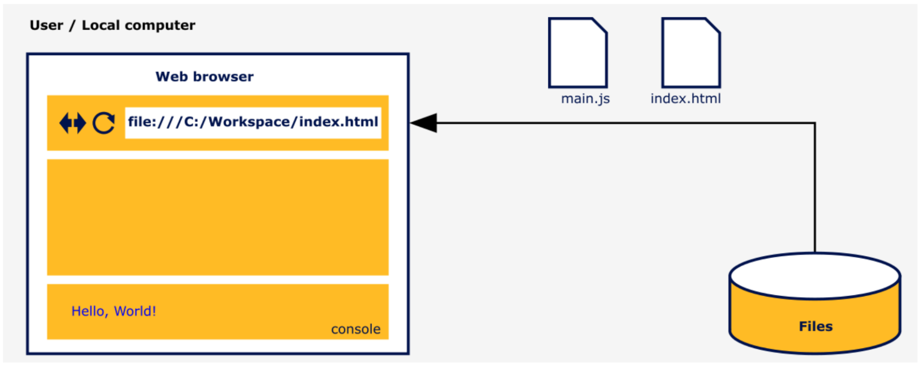
- <head> tag : additional info about document
- JS Code to be executed by browser on page must be attached to HTML using
    - <script> tag : recommended when code is short
    - "src" attribute using "defer" (script should be executed after whole page is loaded) or "async"(script will be executed immediately, but in parallel to parsing the rest of page) attributes

Running JS Code



```
index.html
- - - - - - - - - - - - - - - - -
<html>

...

  <script>
  console.log("Hello, World!");
  </script>

...

</html>
```

Local



Module 2 - Variables, Data Types, Type Casting, & Comments

Declaring Variables

- var or let for variables
- const for constants
    - change in constant causes SyntaxError
        - ex: const greeting = "hello";
- ReferenceError: attempting to print out contents of undeclared variable

Strict Mode

- "use strict" at beginning of code → causes interpreter to use modern JS standards

Hoisting

- allows functions to be used in code before they are declared

Data Types & Type Conversions

- Literals: way of noting specific values

The typeof Operator

- informs about data type
- Possible return values of typeof operator:
    - "undefined" (primitive) → can be returned by typeOf operator
    - "object"
    - "boolean" (primitive)
    - "number" (primitive)
    - "bigint" (primitive) → big written w/ n suffix at end
    - "String" (primitive)
        - \" __________\"
        - String Interpolation: allows you to treat character string as template, in which you can place values
            - ex:
              let country = "Malawi";
              let continent = "Africa";

              let sentence = ' ${country} is located in ${continent}.';
        - WORD.charAt(index)
        - length
        - slice(beginIndex (included), [optional] endIndex (excluded);
        - split(separator, [optional] limit)
    - "symbol" (primitive)
    - "function"
- Conversions
    - through functions like String(), Number(), Boolean()

- Objects
    - record: collection of named fields, each field has its own name(or key) & value assigned to it
      ex:
      ```
      let user1 = {
          name: "Allison",
          surname: "Hart",
          age: 17,
          email: "alli.cheng80@gmail.com"
      };
      ```
        - can 'delete' field
- Arrays
    - instanceof operator: make sure that variable contains an array
    - length
    - indexOf
    - push
    - unshift: push, but added to beginning
    - pop: element w/ largest index return, array length reduced by 1
    - shift: pop, but remove element from beginning, array length reduced by 1, all other elements shifted to left
    - reverse: remove element from beginning (w/ index 0), removed element returned by method, all other elements shifted to left
    - slice: create new array from selected elements of original array
        - 1 argument larger than 0 → [argument, end of array]
        - 2 arguments larger than 0 → [1, 3)
        - 2 arguments, 1st positive, 2nd negative → argument -3 means don't copy last 3 elements
    - concat method: creates new array by attaching elements from array given as argument to original array elements, method changes neither original array nor array specified as argument

Comments
- Single-Line: command + /

Module 3 - Operators & User Operation
Operators
- Assignment Operator: =
- Arithmetic Operators: +, -, *, /, %, ** (power)
- Unary Arithmetic Operators: +______, - _____
- Unary Increment & Decrement Operators: ++, --

- Compound Assignment Operators: +=, etc
- Logical Operators: conjunction (&&), alternative (||), negation (! not symbol)
- Others:
    - typeOf: unary operator that checks type of operand, returns string w/ type name, like "boolean" or "number"
    - instanceOf: binary operator that checks whether an object (left operand) is of some type (right operand), returns true or false, weather variable contains an array
    - delete
    - ternary:
        - ex:
          ```
          console.log(true ? "Alice" : "Bob"); // -> Alice
          console.log(false ? "Alice" : "Bob"); // -> Bob
          ```
        - ex:
          ```
          let name = 1 > 2 ? "Alice" : "Bob";
          console.log(name); // -> Bob
          ```

Dialog Boxes

- Alert Dialog Box
    - alert(" ");
    - window.alert(" ");
- Confirm Dialog Box
    - confirm(" ");
    - window.confirm(" ");
- Prompt Dialog Box
    - prompt(" ");
    - window.prompt(" ");
    - ex:
      ```
      let name = window.prompt("What is your name?", "John Doe");
      name = name ? name : "anonymous";
      let age = prompt("Hello " + name + " how old are you?");
      alert(name + " is " + age + " years old");
      ```

Module 4 - Control Flow - Conditional Execution & Loops

Switch Case Statement

ex:
```
switch(expression) {
        case first_match:
                code
                break;
        ...
        default: code
}
```

For-Of Loop

- loop for arrays

  ex:

```
let values = [10, 30, 50, 100];
let sum = 0;

for (let number of values) {
   sum += number;
}
```
```
console.log(sum); // -> 190
```

For-In Loop

- iterates through all fields of indicated object

  ex:

```
let user = {
   name: "Calvin",
   surname: "Hart",
   age: 66,
   email: "CalvinMHart@teleworm.us"
};

for (let key in user) {
   console.log(key); // -> name, surname, age, email
};
```

Module 5 - Functions

Shadowing

- parameters are treated inside function as local variables

Recursion

regular:

```
function factorial (n) {
   let result = 1;
   while (n > 1) {
      result *= n;
      n--;
   }
   return result;
}
console.log(factorial(6)); // -> 720
```

ex:

```
function factorial (n) {
   return n > 1 ? n * factorial(n - 1) : 1;
}
console.log(factorial(6)); // -> 720
```

Synchronous Callbacks

- instructions executed in order in which they are placed in code

Asynchronous Callbacks

- dependent on particular programming language & on environment
  ex:
  setTimeout Function → delayed action
  setInterval Function → similar to setTimeout

Arrow Functions

- shorter form of function expression
  ex:
  ```
  let add = (a, b) => {
     return a + b;
  }
  console.log(add(10, 20)); // -> 30
  ```

Module 6 - Errors, Exceptions, Debugging, & Troubleshooting

Natural Languages & Communication Errors

- Syntax Error
- Semantic Error: in JS, the interpreter will start program & stop execution after reaching instruction
- Reference Error: when trying to access function or variable that doesn't exist
  - JS engine doesn't know meaning of given name, so it is a semantic error (run-time errors)
- TypeError: when certain value is not of the expected type (run-time error)
- RangeError: when you pass a value to a function that is outside its acceptable range (run-time error)

Error w/out Exceptions

- Arithmetic Errors
  - ex:
    - `console.log(100 / 0); // -> Infinity`
    - `console.log(100 * "2"); // -> 200`
    - `console.log(100 * "abc"); // -> NaN`

Conditional Exception Handling

- instanceof operator used to react differently to specific types of errors

ex:
variable instanceof type
Finally Statement

- last optional block of the try statement

Step-By-Step Program Execution

- resume/continue
- step into

- step over
- step out

How To Deal w/out the Debugger Statement

- clicking resume will cause program to resume execution & stop at 2nd breakpoint

Call Stack

- keep track of its place in script that calls multiple functions

Measuring Code Execution Time

- console.time: start the time measurement
- console.timeEnd: end the measurement & result displayed on console at this point