Intro

- Originally LiveScript in the NetScape browser
- Programs run by an interpreter built into the user's web browser
- Adds interactivity to static content
- Inserted using the `<script>` tag
- `Ctrl+Shift+J` to debug JS on the browser
- Interpreted language - each line treated as input

Pros

- More dynamic HTML pages without web frameworks like Flask

Cons

- Requires Javascript enabled browser
- Requires client to trust the server to run scripts
- Some protection in place but can still cause security issues

Process

1. Request sent to web server
2. HTML page sent to client
3. Javascript executed on client
4. Browser displays output

Syntax

```
<script src="javascript.js">      <!--- for inserting js from external file --->
```

```
// single line comment
/* multiline comment:
JavaScript automatically inserts semicolons at the end of lines. If you have multiple
statements on a single line then you need to add the semicolons. */
```

- First character of variable: alphabet, _ , $
- Variable can be declared using the following:
  - var
    - Scope: Function/Global
    - Can be reassigned
    - Can be redeclared
  - let
    - Scope: Block

        - Can be reassigned
        - Cannot be redeclared
    - const
        - Scope: Block
        - Cannot be reassigned
        - Cannot be redeclared
    - Use without declaration: global scope
- Loosely typed: no need to declare variables or datatypes

Objects

Almost everything is an object

- Booleans, numbers, strings, date, array: object with new keyword
- Primitive datatypes -> not objects

Functions

- Subprogram designed to perform a particular task

```
function name(parameters){
        statements
        // optional return statement
}
```

You can use Function Expressions to store any function, including anonymous ones:

```
let name = function(parameters){
        statements
}

let name = (parameters)=>{
        statements
}

// calling a function
[retval = ] function_name([argument [, ... ]])
```

- Even if the function call does not match the number of args in the definition, the call is still processed
- Function declarations -> hoisted
  Function expressions -> not hoisted
- Variable declarations -> hoisted with an initial value of `undefined`
- let/const -> not hoisted
  You can also pass parameters to a function that does not have any in its definition, or pass a dynamic number of arguments like so:

```
function add()
{
        let sum = 0;
        for(let i in arguments)
        {
                sum+=arguments[i];
        }
        return sum;
}

function add( ...args)
{
        let sum = 0;
        for(let i in args)
        {
                sum+=args[i];
        }
        return sum;
}
```

Note - Hoisting

Default JS mechanism whereby variable and function declarations are moved to the top of their scope before execution; allows you to use any variable undeclared.

- Function declarations -> hoisted
- Function expressions -> not hoisted
- Variable declarations -> hoisted with an initial value of `undefined`
- let/const -> not hoisted

```
function draw(char,i,j)
{
        if(char==undefined)
                char="*";
        if(i==undefined)
                i=3;
        if(j==undefined)
                j=3;
        for(let a=0;a<=i;a++)
        {
                for(let b=0;b<=j;b++)
                        document.write("<p>"+char+" "+"</p>");
                document.write("<br/>");
        }
}
```