*// Lecture: Variables*

script tag in html or with a src link that points to a js file

<script src='script.js'></script>

console.log

**console**.log('hello world');

then console.log a variable

var name **=** 'Kohl';

**console**.log(name);

Kohl

*// Lecture: Variables 2*

PRIMITIVE JS DATA TYPES

NUMBER: Floating point numbers for decimals and integers

STRING: Sequence of characters, used for text

BOOLEAN: Logical data type that can only be true or false

UNDEFINED: Data type of a variable which does not have a value yet.

NULL: Also means 'not-existent'

*//single line comments*

*/\* multi-line*

*comments \*/*

JS knows when to combine datatypes

var name **=** 'Kohl';

var age **=** 31;

**console**.log(name **+** age);

Kohl31

**console**.log(age **+** age);

62

we can have two variables listed

var job, isMarried;

Script reads from top to bottom so if you define a var and do something like console.log it then if you redefine it further down the page and console.log it again you will get two different logs as you go down the page.

Like Console.log we can use

prompt and alert

(both of these don’t show in a console but rather in a pop up)

ar lastName **=** prompt('What is the last name? ')

**console**.log(lastName);

alert(name **+** ' is a ' **+** age **+** ' years old ' **+** job **+** '. Is he married? ' **+** isMarried **+** ".")

*// Lecture: Operators*

Operator precedence determines the way in which operators are parsed with respect to each other. Operators with higher precedence become the operands of operators with lower precedence.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence>

(look to the table in the site for hagiarchy)

*// Lecture: if/else statements*

Pretty explanatory

*// Lecture: Boolean logic and switch*

AND (&&) => true if ALL are true

OR (| |) => true if ONE is true

NOT(!) => inverts true/false value

var age **=** 16;

age **>=** 20; *// => false*

age **<** 30; *// => true*

**!**(age **<** 30); *// => false*

age **>=** 20 **&&** age **<** 30; *// => false*

age **>=** 20 **||** age **<** 30; *// => true*

*CODING CHALLENGE*

*// Lecture: Functions*

When we create functions first write functions then give it a title then give it arguments. Arguments are information you want to pass into the function.

function CalculateAge(yearOfBirth)

then curly braces for the function body which can contain as many lines of code as we wish

so for example we can put in

var age **=** 2018 **–** yearOfBirth

the yearOfBirth is the information we pass into the function when we call the function

so when the function is ready to be run we can call the function from out code and it will return the data that we want it to return. Which in this case will be the age. And we do this by putting the return statement and say that we want to return the age.

**return** age;

so if we did

CalculateAge(1990)

We would get the age calculated but the value wouldn’t be stored anywhere so we create a variable for it.

var ageJohn **=** CalculateAge(1990);

so in var ageJohn = CalculateAge(1998); CalculateAge(1998); calls the function and executes the function with a argument of 1998 . then that value of 1998 is passed into the function and is used inside of the function body

function CalculateAge(yearOfBirth) {

var age **=** 2018 **-** yearOfBirth;

**return** age;

so if we then console.log(ageJohn) we will see the result in the console as 28

functions can also call other functions

our new function is to calculate the years until retirement so we can pass parameters name and year into the function.

function yearsUntilRetirement(name, year)

and then create a variable that will use(CALL) the original function to calculate the age of someone

var age **=** CalculateAge(year);

then a variable to calculate time of retirement minus age

var retirement **=** 65 **-** age;

then to return it we can console log retirement

**console**.log(retirement);

then to make it prettier

**console**.log(name **+** ' retires in ' **+** retirement **+** ' years.');

and then we call the function and pass in the parameters

yearsUntilRetirement('John', 1990);

we can then add more people and their birth years

yearsUntilRetirement('Mike', 1969);

yearsUntilRetirement('Mary', 1948);

however we now see that Mary has already retired because the console log said retired in -5 years.

So we need to add an if statement to say if their years to retirement is less that or equal to zero we will console log the info we had before but if not (else) we will print “has already retired”

**if** (retirement **>=** 0){

**console**.log(name **+** ' retires in ' **+** retirement **+** ' years.');

} **else** {

**console**.log(name **+** ' has already retired.');

}

So to overview we had parameters inside the function

name, yearOfBirth

and we defined the values

'John', 1990

And called them so they were placed(passed) inside the original function as arguments

*// Lecture: Statements and expressions*

So far we have declared functions like this

function whatDoYouDo(job, firstName) {}

Now When we write function expressions we start by writing a function without a name and then pass in an argument list

function (job, firstName);

then you assign this function to a variable.

var whatDoYouDo **=** function (job, firstName) {

we are going to practice some switch statements

**switch**(job) {

**case** 'teacher':

**return** firstName **+** ' teaches kids how to code';

if the case is teacher we will return the statement and nothing else because a return will stop the function. And return to the place where the function was called.

And then call it

whatDoYouDo('teacher', 'John');

but in this case we are just going to conslole.log it

and it should log the string John teaches kids how to code

we then can make more statements by putting setting up more arguments.

So JS expressions are pieces of code that always produce a value

If you would wright 3+2 in the console it would return 5 and 5 would be the result

That’s an expression.

Same is when we call a function like the console

whatDoYouDo('teacher', 'John')

so when we call it, it returns a value. So the function call is an expression



Anything that we do that produces a result is a expression

Even

is an expression

So when JS expects a value we always have to wright an expression

Statements however just perform actions. So they do things but they don’t necessarily produce immediate results. Ex. If else statements, or a while loop, or even a function declaration don’t produce any immediate values and so they are called statements.

the console log produces its result but the statement doesn’t produce anything

So coming back to functions, we learned that function expressions produce an immediate result while function declarations do not.

*// Lecture: Arrays*

Can be done like this

var years **=** **new** Array(1990, 1969, 1948);

but mostly done like this

var names **=** ['John', 'Mark', 'Jane'];

arrays are 0 base or start at 0 instead of 1. Each unit in a array is called a indicy, they are indices. Each indacy value has an index. Ex. the index of 1 would be Mark. You call also call it by saying John is the first indacy. The arrays index of 0 is John

**console**.log(names[0]);

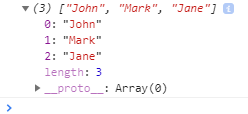


You can also log the whole array

**console**.log(names);



expanded



We can also access the number of indices by logging names.length

**console**.log(names.**length**);



We can also mutate the data in the array by setting additional data

names[1] **=** 'Ben';

names[5] **=** 'Mary';

**console**.log(names);



If we wanted to add marry as the last index of the array we can use the property names.length which equaled 3 and insert it in which will make Mary the index of 3 and then the new length of the array will have 4 indexes

names[names.**length**] **=** 'Mary';

**console**.log(names);



So basically, that just added Mary as the last element in the array

Its possible to have multiple data types in arrays

var john **=** ['John', 'Smith', 1990, 'teacher', false];

later their will be functions that we can apply to an array that are called methods. (that are specific to arrays)

john.push()

which is like a regular function and .push is a function OR PUSH METHOD that will add an element at the end of the array

var john **=** ['John', 'Smith', 1990, 'teacher', false];

john.push('blue');

**console**.log(john);



Another method for the array is .unshift which is like .push except instead of adding it to the end of the array it adds it to the beginning of the array.

john.unshift('Mr.')



John.pop removes a element from the end

var john **=** ['John', 'Smith', 1990, 'teacher', false];

john.push('blue');

john.unshift('Mr.');

john.pop();

**console**.log(john);



(nolonger blue element on the end of the array)

.shift method removes an element from the beginning of the array

var john **=** ['John', 'Smith', 1990, 'teacher', false];

john.push('blue');

john.unshift('Mr.');

john.pop();

john.shift();

**console**.log(john);



.indexOf finds which number index the element passed into it is in the array

**console**.log(john.indexOf(1990));



If the element is actually not present in the array then it will return -1 in the console.log

**console**.log(john.indexOf(22));



So we can test if something is in the array by

john.indexOf('designer') **===** **-**1

**?** */\*(? = then)\*/* 'John is NOT a designer'

**:** */\* : = and else \*/* 'John is a designer'

However this doesn’t work because this isn’t assigned to any variable.

We say var isDesigner and then log it to the console

var isDesigner **=** john.indexOf('designer') **===** **-**1

**?** */\*(? = then)\*/* 'John is NOT a designer'

**:** */\* : = and else \*/* 'John is a designer';

**console**.log(isDesigner);



(this ? and : is an in-line if statement. Just shorter syntax)