**Javascript**

**Program**

* Referred to as the source code / code.
* Set of instructions to tell the computer what tasks it needs to perform.
* Syntax: Format and combination rules are called computer language or Syntax.

**Statements**

* A group of words, numbers and operators that performs a specific task is a statement.
* “Statements are a full sentence.”
* Variables: Symbolic placeholders for the values themselves.
* Literal Value: A value by itself, it stands alone without being stored in a variable.
* Operators: +, -, =, etc. perform actions with the values and variables such as assignment and mathematic multiplication.

**Expression**

* A statement is made up of one or more expressions.
* “Expressions are sentence fragments”
* Any reference to a variable or value, or a set of variable(s) and value(s) combined with operators.
* Literal Value Expression: Has just the literal value
* Variable Expression: Retrieves the current value of the variable.
* Arithmetic Expression: Performs a mathematical function on the expression
* Assignment Expression: Assigns something to the variable.

**Executing a Program**

* Executing a program: Running a program
  + A special utility on the computer is used to translate the code we write into command a computer can understand.
  + Interpreting the code: Translation that occurs every time the code is run, from top to bottom, line by line.
  + Compiling: Translation is done ahead of time, so when it’s run it’s already ready to go.
  + JavaScript is compiled on the fly and then immediately run.
* Output: What displays on the console.
  + Printing: console.log( x);
  + Object reference: console.
  + Function call: log(b), alert
* Input: Usually HTML elements have text boxes that accept data for JS to handle.
  + Prompt( x ); function
    - X: A string value that you’d like displayed above the text box.
  + You can store this as a variable.

**Operators**

* Perform actions on variables and values.
* Assignment: =
* Not: !
* Equals: ==, checks for coercion
* Strict Equals: ===, does not allow coercion
* Loose not-equals: !=
* Strict Not-Equals: !==
* Addition: + or +=
* Subtraction: - or -=
* Multiplication: \* or \*=
* Division: / or /=
* Modulo: %, returns the remainder of the two values when divided.
* Increment: ++
* Decrement: --
* Less Than: < or <=
* Greater Than: > or >=
* And: &&
* Or: ||
* Ternary: ( comparisonStatement ) ? x : y
* Binary Operators: Take in two values.
* Unary: Use one value ex: ‘typeof’

**Types**

* How you classify data within a variable.
* Types of Types
* Number: (Falsy when 0, -0, NaN)
  + Infinity: Can also be used
  + -Infinity:
* NaN (Always Falsy)
* String: Represent text
  + Escaping the character: Use newline characters when you have certain characters in your string.
  + Can concatenate strings with +
  + Backticked strings can have template literals inside of them.
  + String methods:
    - .indexOf
    - .trim
    - .padStart(# of spaces, character)
    - .trim
    - .repeat(# of times)
* Boolean
  + Uppercase letters are always less than lowercase letters.
  + NaN is NOT equal to itself.
* Undefined (Always Falsy)
* Objects
* Null (Always Falsy)
* False (Always Falsy)
* Literals: Values that are included directly in the source code.

Converting Between Types

* Coercion: Converting between data types.
* Number (“X”);
  + Converts a string into a number.
* Implicit coercion: When you try to compare two values of different types.

**Best Practices Writing**

* Have variable names that make sense.
* Have comments where it’s necessary.
  + Comments should explain why, not what.
* Single-line comment: // comment //
* Multiline comment: /\* comment\*/

**Variables**

* A symbolic container that holds a value.
* Also known as a binding, use the keyword let.
* Use Let and const to define a variable.
* Can define multiple variables on one line (let one =1, two = 2)
* Const: Constants a variable that doesn’t change throughout your program.
  + Variable should be written all capital with underscores(\_) for spaces
* Keywords:
  + break case catch class const continue debugger default delete do else enum export extends false finally for function if implements import interface in instanceof let new package private protected public return static super switch this throw true try typeof var void while with yield

**Blocks**

* A group of a series of statements
* Defined with a pair of curly brackets { }

**Control Flow Statements**

* If
  + If you pass something that isn’t a Boolean, the type may be coerced.
  + Falsely: 0 & “”
* Else IF
* Else
* Switch Statement
* While Loop
  + Can use break to stop the loops.
* Do While Loop
* For loop

**Function**

* A named section of code that can be “called” by name and the code is run each time.
* Functions can take arguments (parameters, values you pass in) and can return a value back.
* Structures programs and reduced repetition.
* Every time the function is called, new instances of these bindings are created.
* Executing a function is called invoking or calling it.
* Try to make functions serve only one purpose.
* Pure Function: Does have side-effects and also doesn’t rely on side effects from other code.
  + Always produced the same value when called with the same arguments.
* Functions are conceptually moved to the top of their scope.
* Example existing functions:
  + Prompt(“string”);
  + Console.log(“string”);
  + Math.max(#, #, #);
  + Math.min(#, #, #);
* Arrow functions
  + Const x = (parameter1, parameter2) => {
    - Function body;
    - }

|  |  |  |
| --- | --- | --- |
| Const square = function (x) {  return x \* x;  } | Const square = (x) => {return x\*x} | Const square = x => x\*x; |

* If a function has no parameters at all, it’s parameter list is just an empty set of parenthesis. () =>{ body };

**Optional Arguments**

* If you create a function that should be able to accept a varying amount of parameters, make sure to enter a condition within the function if it isn’t able to be defined.

function minus(a, b) {

if (b === undefined) return -a;

else return a - b;

}

* You can also define the parameter in the parenthesis, this will replace an undefined expression.

**Scope: Accessibility of variables**

* Lexical scope: Code in one scope can access variables of either that scope or any scope outside of it.
* Global Scope: Bindings (let’s and const’s that exist outside of any function or block)
* Local Scope: Bindings that are created for function parameters or declared inside a function.

**Values and Types**

JavaScript has typed values not variables.

Can use the “type of” Keyword to determine the type of the value.

* + Types: String, Boolean, Number, Null, Undefined, Object, Symbol

**Closures**

* The ability to create a reference to a specific instance of a local binding in an enclosing scope.

**Recursion**

* Functions can call themselves if they don’t overflow the stack.
* It is typically slower to call a function several times through recursion than to loop.
* Best to use when there are problems that require exploring or processing several “branches”.

**Data Sets**

* A way to represent digital data within our machine’s memoru.
* Objects
  + A compound value where you can set properties
  + Called using dot operations
  + Collection of properties
  + Object.delete(); : Delete the object
  + Object.keys(); :
* Array: Hold data in a fixed sequenced order.
  + Syntax: var arr = [“hello”, 42, true]
  + Multivariable Arrays: Can have an array within an array.
    - Ex: [[76, 9], [4, 11]]

function phi(table) {

return (table[3] \* table[0] - table[2] \* table[1]) /

Math.sqrt((table[2] + table[3]) \*

(table[0] + table[1]) \*

(table[1] + table[3]) \*

(table[0] + table[2]));

}

* + - .includes: Check to see whether or not a value is in the array.
* For Loop
  + For (let entry of Journal) {
    - Statement
    - }
* Methods: Properties that contain functions.
  + .push(x); : Puts x on the end of the array.
  + .pop(); : Removed the end of the array.
  + .assign() : Copies all the properties from one object into another.
  + .shift(): Get and remove the first item off of the array.
  + .unshift(): Put something in position 0 of an array.
  + .indexOf()
  + .lastIndexOf()
  + .slice(x, y): Takes the start and end indicies and returns an array that has only the elements between them. Start is inclusive and the end is exclusive.
  + .concat(): Adds arrays to other arrays.
* Mutability:
  + Numbers, strings and Booleans are all immutable.
  + Objects are mutable, you can change their properties.
  + == Compares identity
* Rest Parameters
  + Used to accept the values of all the arguments given.
  + Function max(…numbers){}
  + Can use the same … notation to call a function with an array of arguments.
* Function Scopes
  + Use the var keyword to declare a variable that will belong to the current function scope or global scope if at the top level.
  + Hoisting
    - Var’s can be hoisted to the top of their scope.
* Math Object
  + .cos, .sin, .tan, .acos, .asin, .atan
  + .random
  + .PI
  + .floor: Rounds down to the nearest whole number.
  + .ceil: Rounds up to the nearest whole number
  + .abs: Absolute value
* JSON
  + Serialize the data into a flat description format so that it can be picked up and read by other programs
  + .stringify(): Converts to JSON
  + .parse(): Converts back to JS.
* Abstraction
  + Abstractions hide details and give us the ability to talk about problems at a higher level.
  + Higher-Order Functions: Functions that operate on other functions.
    - Can take them as parameters or can return them.
    - Allow us to abstract over actions, not just values.
  + You can add arrow functions as parameters of other functions
    - They require both the bracket and the parenthesis at the end.
  + Can create new types of control flow
  + Can Change other functions
  + Pure Function: A function that doesn’t modify the variable it was given.
    - Map
      * Transforms an array by applying a function to all of its elements.
    - Reduce
      * It builds a value by repeatedly taking a single element from the aray and combining it with the current value.
* Object Orientated Programming
  + A set of techniques that use objects as the central principle of program organization.
  + Divide programs into smaller pieces and make each piece responsible for managing its own state.
  + Encapsulation
    - Hiding the inside of a method to abstract having to know the intricacies.
    - Interfaces
      * Limited sets functions or bindings that provide useful functionality at a more abstract level
      * Properties that are part of the interface are called public.
      * The others that outside code should not be touching are private.
    - Use \_ at the start of property names to indicate that those properties are private.
  + Methods: Properties that hold function values
    - [Object.js](file:///Users/vn503o9/Desktop/Udemy/Javascript/Eloquent%20Javascript%20Excersizes/Javascript-Exercises-/Object.js)
    - Can use function.call() method
    - Arrow functions don’t have a “this” function.
      * This is good when you need to use the this from another function.
  + Prototypes
    - Another object that is used as a fallback source of properties.
    - You can use Object.prototype to get the prototype.