JavaScript Training – Codecademy

**Introduction**

* To make a string, use “quotes”
* To make a function, use **function()**
* End with a semicolon
* To set a variable, use **var variableName = variable value**
* Console keyword refers to an object, a collection of data and actions that we can use in our code.
* **Console.log()** will be printed, or logged, to the console
* Single line comment starts with //
* Multi-line comment begins /\* and ends \*/ (can also use to comment in the middle of a line of code)
* Data types
  + Number – 1, 2, 3, etc
  + String surrounded by ‘string’ or “string”
  + Boolean – true or false
  + Null – represents no value, denoted by keyword **null**
  + Undefined – keyword **undefined,**  means no value, but different than **null**
  + Symbol – unique identifier
  + Object – collections of related data
* Arithmetic Operators
  + Add: **+**
  + Subtract: **-**
  + Multiply: **\***
  + Divide: **/**
  + Remainder: **%**
  + When these are put in parentheses, will complete the function
* String Concatenation
  + **+**  can be used to join two types of string data
* Properties
  + String instances have a property called **length**, measure the number of characters in the string.
  + The **.**  operator.
* Methods
  + Methods are actions we can perform
    - The **.**
    - The method name
    - Opening and closing parentheses
    - **.toUpperCase()** (changes to all upper case)
    - **.startsWith()** (shows true or false)
* Built in Objects
  + **Math**  - more complex math operations
* Variables
  + A variable is a container for a value. They label and store data in memory
  + You can create variables with a descriptive name, store or update variable information, or reference or get the information stored in the variable.
  + **Var** – used to create or declare a new variable
  + Camel Casing – grouping words, first is lowercase, then subsequent words have first letter capitalized.
  + **=** is the assignment operator
  + Variable names can’t start with numbers
  + Variable names are case sensitive
  + Variable names can’t be the same as keywords
* Another variable declaration is **Let**
  + A variable can be declared without being assigned a value
* Another variable declaration is **const**
  + Variables declared like this can’t be changed
  + Variable must be assigned a value when declared
* Additional operators
  + **+=** reassigns variable and adds next value after the operator
  + **-=** reassigns variable and subtracts next value after the operator
  + **\*=** reassigns variable and multiplies next value after the operator
  + **/=** reassigns variable and divides next value after the operator
  + **++** increases the value of a variable by 1
  + **–** decreases the value of a variable by 1
* String concatenation with variables
  + **+**  is used to combine two string values, even if they are stored in variables
* String Interpolation
  + Used to insert variables into strings.
  + Backticks are template literals **``**
  + ${variable} is a placeholder
* Typeof operator
  + Used to keep track of the data types of variables
  + **Typeof** checks the value to its right and returns the data type
* ES6 made the following changes to JavaScript
  + New keywords let and const
  + New function syntax using Arrow functions
  + Creation of Classes
  + Parameters with default values
  + Promises for asynchronous actions

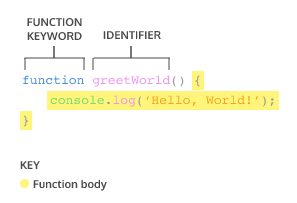
**Conditional Statements**

* Conditional statements are if-else statements.
* A conditional statement checks a condition and performs a task based on the condition status.
* The **If** statement is composed of **if (blank) {code block}**
* The **If/Else** statement
  + If (true/false)
    - {code block}
  + Else
    - {}
* Comparison Operators
  + Comparison operators compare the value on the left with the value on the right
  + Less than: **<**
  + Greater than: **>**
  + Less than or equal to: **<=**
  + Greater than or equal to: **>=**
  + Is equal to: **===**
  + Is not equal to: **!==**
  + all comparison statement evaluate to either true or false
* Logical Operators
  + Logical operators work with Boolean (true/false) values
  + The **and** operator: **&&**
    - When using &&, both conditions must evaluate to true
  + The **or** operator: **||**
    - Used when only one condition needs to be true
  + The **not** operator: **!**
    - The ! reverses the value of a boolean
* True/False with strings and numbers
  + If(variable) = true, because a variable value does exist
  + False condition
    - 0
    - Empty strings
    - Null
    - Undefined
    - NaN
* Ternary Operator
  + isNightTime ? console.log(‘Turn on the lights!’) : console.log(‘turn off the lights!’)
  + isNightTime is the condition
  + blue is the first expression
  + green is the second expression
  + if the condition is true, first expression executes
  + if condition is false, second expression executes
  + **? is between the condition and statements**
  + **: is between the two expressions**
* Else if statements
  + The **else if** statement is used to add more conditions to an **if else** statement.
* Switch statements
  + **Switch () {**
    - **Case :**
      * **code**
      * **Break**
    - **Case :**
      * **code**
      * **Break**
    - **Case :**
      * **code**
      * **Break**

**}**

**Functions**

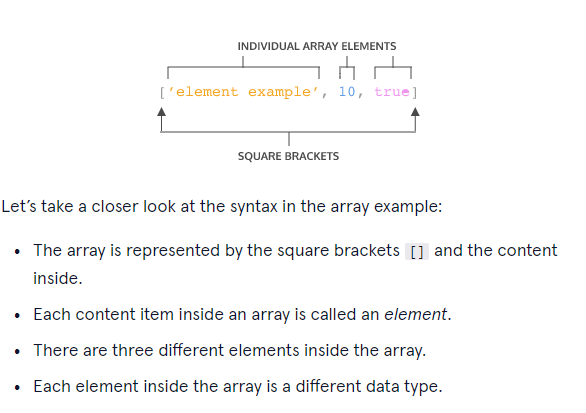
<https://www.youtube.com/watch?v=cKbN6-1ZQ3A>

* A function is a reusable block of code that groups statements to perform a task
* Declare a function by saying **function**
* <https://www.youtube.com/watch?v=cKbN6-1ZQ3A>
* Functions can be called as many times as needed
* Functions have parameters and arguments
  + Parameters specify the values that can be contained in the function
  + Arguments are the values that are passed to the function
* Default parameters allow a function to have a predetermined value in case there is no argument or an undefined argument
* Using the keyword **return** makes the function return a value
* Helper function is a functions returned value used inside another function
* Another way to define a function is a function expression
* Arrow functions are a shorter way to write functions **() =>**
* Parentheses are needed for functions if there are 0 or multiple parameters

**Scope**

* Scope defines where variables can be accessed or referenced
* Block is the code found within a set of curly braces {}
* Global scope means variables are declared outside of blocks. They can be accessed by any code in the program.
* Block scope means that variable is only accessible to the code within the braces
* Having too many variables lead to scope pollution, accidentally reassigning variables

**Arrays**

* Arrays are used to organize and store data
* An array literal creates an array by wrapping items in square brackets **[]**
* 
* Each element in an array has a numbered position known as its index.
* Arrays in JavaScript are zero-indexed, meaning positions start with 0, rather than 1.
* Variables declared with let can be reassigned
* Variables with const can’t be reassigned; however, elements in an array declared by const can still be changed.
* **.length** is part of an arrays properties. Can be used to determine how many items are in an array
* **.push** is used to add items to the end of an array
* **.pop** removes the last item of an array
* Additional array methods are
  + **.join()** – creates and returns a new string by concatenating all the elements in an array
  + **.slice()** – returns a shallow copy of a portion of an array into a new array object
  + **.splice()** – used to remove, replace or add new elements
  + **.shift()** – removes the first element and returns the removed element
  + **.unshift()** – adds elements to the beginning of the array
  + **.concat()** – used to merge two or more arrays
  + **.indexOf()** – used to locate the position of an element in an array

**Loops**

* A programming tool that repeats a set of instructions until a stopping condition is reached
* The **For**  loop contains three expressions
  + An initialization that starts the loop and can be used to declare the iterator variable
  + A stopping condition
  + An iteration statement to update the iterator variable on each loop
* Nested loops
  + For()
    - For()
* The **While** loop: continues in a loop while a condition is true
* The **Do…While** statement says to do a task once and then keep doing it until a specified condition is no longer met
* The **break** keyword allows programs to break out of the loop from within the loop’s block

**Higher-order Functions**

* A function can be assigned to a reference variable
* Functions are first class objects
* A higher order function is one that accepts functions as parameters, returns a function, or both.
* Functions that go into higher order functions are called callback functions
* When using a callback function, we are calling the code of the function, not the result.
* Iterators are methods called on arrays to manipulate elements and return values
* Iterator methods
  + The **.forEach()** will execute the same code for each element of an array.
  + forEach() and other iterator methods have a callback function as its return. The return value for .forEach() will be undefined if not defined in another function.
  + The **.map()** takes an argument of a callback function and returns a new array.
  + The **.filter()** returns a new array after filtering out certain elements from the original array.
  + The **.findIndex()** locates the index location of the first element in an array that qualifies as true.
  + The **.reduce()** method returns a single value after iterating through the elements of an array.

**Objects**

* The seven fundamental data types are
  + String
  + Number
  + Boolean
  + Null
  + Undefined
  + Symbol
  + Object
* JavaScript objects are containers storing related data and functionality.
* Objects can be assigned to variables by **let variable = {}**
* An object is filled with unordered data, which is organized into key-value pairs.
* A key is like a variable name that points to a location in memory that holds a value.
* 
* If a key is string and contains no special values, eg space or punctuation, no quotes are needed.
* Properties can be accessed through dot notation.
* Properties can also be access through bracket notation []
* Bracket notation but be used when accessing keys that have numbers, spaces or special characters in them.
* Variables can also be used in brackets to select the keys of an object, especially when working with functions.
* Properties can be added to an object after it is created
* When data is stored on an object, is it called a method.
* A property is what an object has, and a method is what an object does.
* To invoke an object method, use object.method()
* Objects can be nested
* MAKE SURE TO PUT COLONS AND COMMAS IN THE RIGHT PLACE
* Objects are passed by reference
* The **for….in** function will execute a block of code for each property in an object.
* The **this** keyword references the calling object, which provides access to the calling object’s properties.
* Don’t use arrow functions when using **this**  method
* Using **\_** before an object denotes that the property should not be altered
* Altering an object property can cause functions to return altered or undefined values.
* Getters are method that get and return the internal properties of an object.
* Properties can’t share the same name as the getter/setter functions
* Setters reassign values of existing properties within an object.
* Factory functions is a function that returns an object and be reused to make multiple object instances.
* Key-value pairs can be destructured in factory functions by rephrasing:
  + Name: name, = name,
  + Age: age, = age,
* Built-in Object Methods – programmed methods that can be used for objects
* Make sure to capitalize O in the Object.method() methods.

**Modules**

* Modules are reusable pieces of code that can be used and then exported to use in another file
* Use **module.exports** to export the module for use in the code.
* The **require()** function accepts a string as an argument. That string provides the file path to the module you would like to import.

**Asynchronous programming**

* The **promise** object is used to help handle asynchronous programming.
* Promise is used like **new promise(executorFunction)**
* Executor function has two parameters, **resolve, reject**
* The **setTimeout()** function is a node API used to callback functions to be performed after a delay.
* The initial state of an asynchronous promise is pending.
* The **.then()** function is used with two handles, onFulfilled and onRejected.
* The **.catch()** function takes on argument, onRejected.
* Chaining promises together is called composition.
* The **promise.all()** function is used to run multiple promises concurrently.
* The promise.all() accepts an array of promises and returns a single one.
* The **async()** keyword is used to write functions that handle asynchronous actions, and always return a promise.
* The **await** keyword can only be used inside the async function. It is an operator, it returns the resolved value of a promise.

**HTTP requests**

* HTTP stands for HyperText Transfer Protocol
* The transfer of info happens using TCP (Transmission Control Protocol)
* The **XMLHttpRequest** is used in JavaScript to interact with servers.