JS 2 puporses

1. Programmatically alter web pages based on user actions
   1. Requires an API through which they can manipulate the structure and appearance of the HTML page
   2. DOM (Document Object Model) API
2. Exchange messages with a server over a network
   1. XHR (XMLHttpRequest) interface and Fetch API let you communicate with a server

Node.js

* Node.js is a runtime environment that turn JS into a general purpose programming language that can run apps on almost system
* Creators of Node.js took the open-source Chrome V8 JavaScript engine and added APIs and tools required for desktop and server computer
* General purpose enviro like Node.js requires:
  + Ability to read and write disk files (disk I/O)
  + Ability to read and write via terminal (standard I/O)
  + Ability to send and receive messages over a network (network I/O)
  + Ability to interact with a database
* Node.js has API’s and packages for all these tasks and more
* It also provides an interactive REPL (read-eval-print loop) where you can execute JS commands and get instant results
* Provides tools for debugging and inspecting programs at runtime
  + Though not easy, generally need to use a browser
* Adobe’s Acrobat supports JavaScript to automate and animate elements in a document
* npm node’s package dependency manager

The Command Line

* This section is for commands needed to run and test JS code from CL

The Node REPL

* Read-eval-print loop for a programming language is an interactive enviro where you can type commands and expression in that language and get immediate results
* Node.js comes with one REPL, which youcan access with the node command:
  + node
* node shows return values with a dimmer appearance

Documentation

* JS docs focus on available data types and how to use them
* Sometimes referred to as API
  + Confusing bc we know API also refers how apps talk to each other
* Programming enviro provides two main types of reusable code to an app:

1. Components and operations part of the core language aka standard library
2. Components and operations specific to a runtime environment

* 3 most crucial items:

1. Constructors and Name Types
   * Ex. String. That’s the name of the constructor for the data type that the page describes
   * Constructors are factories that create values of a particular data type
   * They provide the blueprint for a data type with the same name
   * The string constructor, for example, is a factory that creates values of type string
2. Method name
   * Methods are functions that need a value to use to call the function
   * Two types:
     + Instance methods
       - Constructor.prototype.methodName()
     + Static methods
       - Constructor.methodName()
3. Property Name
   * Some data types have properties associated with them
   * Ex. Verb vs noun. Data type’s property is a noun, an operation is a verb
   * Property says something about the value & operation does something with the value
   * You can access the property by appending a dot and the name of the property to that value

* Instance Methods vs Static Methods
  + Instance methods are applied to a value that the constructor represents
    - Ex. ‘Hello, ‘.concat(‘Bob!”)
    - >’Hello, Bob!’
  + Static methods are applied to the constructor itself:
    - String.fromCharCode(97)
    - ‘a’
* Best way to find documentation is search for “mdn”

Data Types

JS has 5 primitive data types:

* String
  + \ tells the computer the next character is part of the string
  + ` ~ ‘ string interpolaton allows you to merge JS expressions with strings
    - `Blah ${expression} blah`
    - `5 plus 5 equals ${5+5}`
* Number
* Undefined
  + Absence of value
  + Console.log() is an example of a function that returns undefined
* Boolean
* Null
  + Represents intentional absence of a value
  + Let foo = null

Type of Operator

* Every value has a data type
* typeof 1, returns ‘number’
* typeof null equals ‘object’

Equality Comparison

* determine whether two values are identical
* ===, avoid == now

String Concatenation

* ‘1’ + 2 = ‘12’
* ‘5’ – 3 = 2

NaN

* 0/0 returns NaN
* Undefined mathematical operations
  + dividing 0/0
  + sqrt(-#)
* trying to convet a non-number value, ‘hello’ to a number

Explicit Coercion

* number(‘1’) = 1
* parseInt()
  + coerce’s strings to numbers
  + parseInt(‘12xyz’) = 12
  + parseInt(‘3.1415’) = 3
  + if too big, infinity or -infinity (>300 digits)
  + parseFloat(’12.5foo’) = 12.5
* Coerce numbers to strings
  + String(12) = ‘12’

Data Structures

* 2 Most common
  + Arrays
    - Lists
    - Array literals (representations of arrays) use brackets
  + Objects
    - Dictionaries
    - Key-value pairs
    - Objects can have multiple elements
    - ({dog: 'barks', cat:'meows'})['cat'] = ‘meows’
      * Need to use paren but not important bc rarely use [] with a literal object

Expressions

* Anything that JavaScript can evaluate to a value, even if undefined or null
* Almost everything is an expression
* Printing/logging vs returning values
  + Ex. Console.log(‘howdy”)
    - howdy = logged
    - undefined = returned value
* Statements
  + Full list on MDN
  + Categories: control flow, declarations, functions and classes, iterations, others
  + Ex variable declaration

**Variables**

Variables store info that a program must reference or manipulate

They provide a way to label data with a descriptive name to help us and other readers understand the program

They are containers that hold information

Declaring and Assigning Variables

* Variable declaration asks JS engine to reserve space for a variable with a name and initialize it with a value
* Preferred method to declare:
  + Let
  + let firstName = ‘Casey’
  + console.log(`Your first name is ${firstName}.`)

Declaring Constants

* const firstName = ‘Casey’
* once initialized, you can’t give it another value
* they remain the same throughout the execution of a program

Variable Scope

* determines where a variable is available in a program
* variables with let or const keywords have block scope
* if variables are undeclared, they become global variables

**Input/Output**

Complete

**Functions**

* Local variable names between paren are called parameters
* All JS function calls evaluate to a value. By default, that value is undefined. When you use a return statement, you can return a specific value from a function
* Default parameters – parameter is set to something in the beginning
* Global variables vs local

Functions Vs Methods

* Method invocation occurs when you prepend a variable name or value followed by a period (.) to a function invocation

Mutating the caller

* Sometimes a method permanently alters the object that invokes the method
  + Mutates the caller
  + i.e. .pop() method permanently removes the last number in the array
* functions can mutate the arguments
  + changing the value of an index in an array via indexing
  + concat does not do that, it leaves the original array intact
* Mutations are concerns with objects and arrays, not primitives (numbers, strings, Booleans, NaN, undefined)
  + Primitives are immutable

Function Composition

* JS uses a function call as an argument to another function

Three ways to define a function

* Call function before declaration
* Save function to variable
* Arrow function

Review

Github

* Git add .
* Git push
* Git push origin master
* For setting up repo first time:
  + Git remote add origin <https>

Exercises – The basics

Read MDN statements

Need to review $PATH & npm

let rlSync = require('readline-sync')

let name = rlSync.question("What's your name?\n")

console.log(`Good morning ${name}!`)

Let name = prompt(“What’s your name?)

Console.log(`Good Morning, ${Casey})

Review Functions Exercises