* // indicates that the entire line is comment
* /\*  \*/ are used to indicate multi-line comments
* JavaScript has seven types of data types
  + undefined
  + null
  + boolean
  + string
  + symbol
  + number
  + object
* Variables - names that are used to refer to specific data that is stored; can change values over time
  + Use the keyword "var" to declare a variable: *var myFirstVariable;*
  + Variable names can be made up of numbers, letters, and $ or \_, but may not contain spaces or start with a number
  + You can assign values to variables, either by directly using values or other variables; examples:
    - *myFirstVariable = 5;*
    - *myFirstVariable = anotherVariable;*
  + It is common to initialize a variable to an initial value in the same line that it is declared; example:
    - var numOfCats = 2;
  + Before variables are given values, they have a value of "undefined"
    - Doing a mathematical operation on an undefined variable will result in NaN (which means "Not a Number")
* Function and variable names are case sensitive
* It is best practice to write variables names in camelCase
* Mathematical operations can be performed on variables
  + Addition operator: +
  + Subtraction operator: -
  + Multiplication operator: \*
  + Division operator: /
  + Remainder operator: %
* Increment a variable - add one to the variable
  + *var1++;* is equivalent to *var1 = var1 + 1;*
* Decrement a variable - subtract one from the variable
  + *var1--;* is equivalent to *var1 = var1 - 1;*
* Decimal numbers are sometimes called floating numbers or floats
* There are operators to allow you to do a mathematical operation and an assignment in one step; for example: +=, -=, \*=, /=
* A string is enclosed in single (') or double (") quotation marks
  + If a string contains a quotation mark, you have to "escape" it by putting a backslash (\) before the quotes - this indicates that the quote is not the end of the string
  + If single quotes are used for indicating a string, then double quotes can be used inside the string without needing to use backslash
  + If double quotes are used for indicating a string, then single quotes can be used inside the string without needing to use backslash
  + Backslash (\) can be used to escape other characters as well:
    - \' (single quote)
    - \" (double quote)
    - \\ (backslash)
    - \n (newline)
    - \r (carriage return)
    - \t (tab)
    - \b (backspace)
    - \f (form feed)
  + Strings can be concatenated (or joined) with other strings using the + operator
    - += can be used to concatenate a string to the end of an existing string
  + .length property can be used to find the length of a string variable or string literal
  + bracket notation can be used to get the character at a specific index within a string
    - First character within a string is at index 0
      * Example: string[0]
    - You can get the last character of a string by subtracting one from the string's length:
      * string[string.length-1]
      * You can also use this notation to find nth character to last
        + Example: 2nd character to last: string[string.length-2]
  + In JavaScript, you cannot change the individual characters of a string variable, but you can replace the entire value of a string variable with another string
* Arrays
  + A type of variable in which we can store different pieces of data that are usually related in some way
  + Example: var vehicles = ["car", "bike", "scooter"];
  + Multi-dimensional arrays - arrays which are nested within other arrays
    - var people = [["Nancy", 23], ["Charley", 45]];
  + You can assign one array to another:
    - var newArray = [people];
  + Indexes can be used to access the data within arrays; starting from position 0

var array = [5,10,15];

array[0]; // equals 5

var data = array[1]; // equals 10

* + Values within arrays can be changed easily:
    - numbers[0] = 45; // sets the first element of array numbers to 45
  + Bracket notation can be used to get the elements within arrays of arrays

var arr = [

  [1,2,3],

  [4,5,6],

  [7,8,9],

  [[10,11,12], 13, 14]

];

arr[3]; // equals [[10,11,12], 13, 14]

arr[3][0]; // equals [10,11,12]

arr[3][0][1]; // equals 11

* + .push() function can be used to append data to the end of an array
    - Example: array.push(10); // 10 is added to the end of the array
  + .pop() function returns the last value of an array (and removes it from the array)
  + .shift() function returns the first value from an array (and removes it from the array)
  + .unshift() function appends the data to the beginning of the array
* Function
  + Declaring a function:

function myFunc() {

}

* + Calling the function:

myFunc();

* Function can accept parameters
* Functions can be used to do calculations
* Example (function accepts two paramters and then the two parameters are added and then output to the console:

function functionWithArgs (param1, param2){

console.log(param1 + param2);

}

//Calling the function:

functionWithArgs (2, 4);

* Variables with Global scope - if a variable is declared outside of a function than it has global scope
  + That variable can be used in any function
* Variables with local scope - variable declared within a function
  + That variable can only be used within that function
* Parameters also have local scope
* You can have a global variable and a local variable with the same name
  + If that happens, the local variable takes precedence over the global variable
* Values can be passed to functions as arguments
* Whatever is to the right of the assignment operator is equalated first and then assigned to the variable on the left
* Justin (sent me invite to Slack), Ian (went to codecamp)
* Queue
  + New items are added to the back and new items are added off the front
* Boolean data type
  + Only two possible values: true or false
* If statements can be used with Boolean values to do certain things when something is true and another thing when something is false
  + Example:

function test (myCondition) {

  if (myCondition) {

     return "It was true";

  }

  return "It was false";

}

* Equality operator (==) compares two values and returns true is they are equivalent and false otherwise
  + Equality operator uses type coercion
* Type Coercion - When comparing two different data types, JavaScript converts one datatype to the other
  + 1 == 1 // true
  + 1 == 2 // false
  + 1 == '1' // true
  + "3" == 3 // true
* Strict equality (===) compares two values but does not perform type conversion
  + 3 === 3 // true
  + 3 === '3' // false
* *typeof* operator returns the type of a variable or value
  + typeof 3 // returns 'number'
  + typeof '3' // returns 'string'
* Inequality operator (!=) is the opposite of the equality operator
  + Will return *false* where equality operator would return *true* and vice versa
  + Converts data types to do the comparison, just like the equality operator
  + 1 != 2 // true
  + 1 != "1" // false
  + 1 != '1' // false
  + 1 != true // false
  + 0 != false // false
* Strict inequality operator (!==) is the logical opposite of the strict equality operator
  + Returns *false* where strict equality operator would return *true* and vice versa
  + Does not convert data types
  + 3 !== 3 // false
  + 3 !== '3' // true
  + 4 !== 3 // true
* Greater than (>) operator
  + Returns true if the number to the left is greater than the number to the right of the operator; otherwise returns false
  + Converts data values while comparing
  + 5 > 3 // true
  + 7 > '3' // true
  + 2 > 3 // false
  + '1' > 9 // false
* Greater than or equal to (>=) operator
  + Returns *true* if the number to the left is greater than or equal to the number of the right of the operator; otherwise returns *false*
  + Converts data types while comparing
  + 6 >= 6 // true
  + 7 >= '3' // true
  + 2 >= 3 // false
  + '7' >= 9 // false
* Less than (<) operator
  + Returns true if the number to the left is less than the number to the right; otherwise returns false
  + Compares data types while comparing
  + 2 < 5 // true
  + '3' < 7 // true
  + 5 < 5 // false
  + 3 < 2 // false
  + '8' < 4 // false
* Less than or equal to (<=) operator
  + Returns true if the value on the left is less than or equal to the value on the right; otherwise returns false
  + Converts data types during comparison
  + 4 <= 5 // true
  + '7' <= 7 // true
  + 5 <= 5 // true
  + 3 <= 2 // false
  + '8' <= 4 // false
* Logical And operator: &&
  + Returns true if both comparisons are true; otherwise returns false
* Logical Or operator: ||
  + Returns true if either operand are true; otherwise returns false
* *else* statements can be combined with *if* statements to return something else when the *if* statement evaluates to *false*
* *else if* statements can be used to chain multiple *if* statements together
* *switch* statements
  + Can be used instead of multiple-if-else statements if you have many options to choose from

function switchOfStuff(val) {

var answer = "";

switch (val) {

case "a":

answer = "apple";

break;

case "b":

answer = "bird";

break;

case "c":

answer = "cat";

break;

default:

answer = "stuff";

break;

}

return answer;

}

console.log(switchOfStuff("b"));

* + *default* statement - executed if there are no matching cases found
  + Multiple cases can be used
* The result of any comparison returns true or false - so you can just return that from a function; instead of having separate if/else statements for comparison being true or false
* A function exists when the first return statement of the function is reached
* Counting Cards exercise
  + Requirements:
    - One parameter: card
    - Increment or decrement global *count* variable
    - Return:
      * A string with current count
      * String "Bet" if count is positive or "Hold' if count is zero or negative
      * Two strings should be separated by a single space
    - Do NOT reset countto 0 when value is 7, 8, or 9.
    - Do NOT return an array.
    - Do NOT include quotes (single or double) in the output.
  + My solution:

var count = 0;

function cc(card) {

  // Only change code below this line

  switch (card) {

    case 2:

    case 3:

    case 4:

    case 5:

    case 6:

      count++;

      break;

    case 10:

    case 'J':

    case 'Q':

    case 'K':

    case 'A':

      count--;

      break;

  }

  if (count > 0) {

    return count + " Bet";

  }

  else {

    return count + " Hold";

  }

  // Only change code above this line

}

// Add/remove calls to test your function.

// Note: Only the last will display

cc(2); cc(3); cc(7); cc('K');

console.log(cc('A'));

* Two ways to access properties of an object
  + dot notation ( . )
    - Example: myCar.door
  + bracket notation ( [ ] ) - need to use this when the property we are trying to access has a space in it's name; can also be used on properties that don't have space
    - Example: myCar["four doors"]
    - If the property was created using quotation marks, you have to use quotation marks when accessing it also
* Accessing the property that is stored in a variable of an object - need to use the bracket notation
  + Use bracket notation without quotes since we are access the value stored in a variable, not the variable itself

// Setup

var testObj = {

12: "Namath",

16: "Montana",

19: "Unitas"

};

// Only change code below this line;

var playerNumber = 16; // Change this Line

var player = testObj[playerNumber];

* You can update properties of an object either through dot notation or through bracket notation

var ourDog = {

  "name": "Camper",

  "legs": 4,

  "tails": 1,

  "friends": ["everything!"]

};

// Changing property

ourDog.name = "Happy Camper"; //or

ourDog["name"] = "Happy Camper";

* You can add new properties to an object in the same way that you modify them
  + Can use either doc notation or bracket notation
  + Don't have to change the object's definition as in other languages
* A property can be deleted from an object, as follows:
  + delete ourDog.bark; // will delete the bark propery
* An object can be used to as a "lookup table" or an array
  + Properties are used to lookup the values of the properties
* All objects have a .hasOwnProperty(propertyname) method, which can be used to check if a method has a particular property
* JavaScript object can be used to handle flexible data - combination of strings, numbers, boolean, arrays, functions, and objects
* JavaScript Object Notation (JSON) - related data interchange format
* Nested objects
  + Objects can be nested inside other objects
  + Sub-properties of objects can be accessed by joining dot and bracket notation together

// Setup

Example:

var myStorage = {

"car": { "inside": {

"glove box": "maps",

"passenger seat": "crumbs"

},

"outside": {

"trunk": "jack"

}

}

};

var gloveBoxContents = myStorage.car.inside["glove box"];

console.log(gloveBoxContents);

* Arrays can be nested other arrays
  + Example of accessing nested arrays:

var secondTree = myPlants[1].list[1];