| **Code** | **Output** |
| --- | --- |
| \' | single quote |
| \" | double quote |
| \\ | backslash |
| \n | newline |
| \r | carriage return |
| \t | tab |
| \b | backspace |
| \f | form feed |

*Note that the backslash itself must be escaped in order to display as a backslash.*

**Find the Length of a String**

You can find the length of a String value by writing .length after the string variable or string literal.

"Alan Peter".length; // 10

For example, if we created a variable var firstName = "Charles", we could find out how long the string "Charles" is by using the firstName.length property.

Basic JavaScript: Manipulate Arrays With pop()

Another way to change the data in an array is with the .pop()function.

.pop()is used to "pop" a value off of the end of an array. We can store this "popped off" value by assigning it to a variable. In other words, .pop()removes the last element from an array and returns that element.

Any type of entry can be "popped" off of an array - numbers, strings, even nested arrays.

var threeArr = [1, 4, 6];  
var oneDown = threeArr.pop();  
console.log(oneDown); // Returns 6  
console.log(threeArr); // Returns [1, 4]

Basic JavaScript: Manipulate Arrays With shift()

pop()always removes the last element of an array. What if you want to remove the first?

That's where .shift()comes in. It works just like .pop(), except it removes the first element instead of the last.

Use the .shift()function to remove the first item from myArray, assigning the "shifted off" value to removedFromMyArray.

## Basic JavaScript: Manipulate Arrays With unshift()

Not only can you shiftelements off of the beginning of an array, you can also unshiftelements to the beginning of an array i.e. add elements in front of the array.

.unshift()works exactly like .push(), but instead of adding the element at the end of the array, unshift()adds the element at the beginning of the array.

IF&Else

If (condition //is true) {

Do whatever code in here

}

var a = 5;

var b = 10;

if (a < b) {

alert("Yes, a is less than b");

} else { alert("Yes, a is equal to b’);

};

if (a == b) {

alert("Yes, a is equal to b");

}

And &&

Or ||

Switch

Ex1

|  |  |
| --- | --- |
| var grade = "Premium";    if ( grade === "Regular") {  alert("It's $3.15");  }  if ( grade === "Premium") {  alert("It's $3.35");  }  if ( grade === "Diesel") {  alert("It's $3.47");  } | var grade = "Regular";    switch (grade) {  case "Regular":  alert("It's a $3.15")  break;  case "Premium":  alert("It's a $3.35")  break;  case "Diesel":  alert("It's a $3.47")  break;  default:  alert("That’s not a valid grade")  }  //defult means if it’s not any od the above cases do the next code |

loops

The while loop is the classic loop in C- based languages, but after you've looked at a few you'll realize that a pattern begins to emerge. That you always have to deal with the same elements. Regardless of what code you put in your loop, whether that's one line or a hundred of them, you're dealing with setting up an index to keep track of the loop and you have to do that outside of the loop itself. Then you're going to be checking the condition and then you have to make sure that you're incrementing the index inside the loop, but at the end of it.

//set up the index

var i = 1;

while ( i < 10){ //check the condition

//do stuff

//do stuff

//do stuff

//do stuff

//etc..

i++ //incresment the index

}

//for loop

//setup-index; //check-Condition; //incresment-index

for (var i = 1; i < 100 ; i++){

//do stuff

//do stuff

//do stuff

//do stuff

//etc..

}

//do--- while loop

var a = 1;

do {

//do stuff

a++;

} while (a < 10);