**Variables in More Detail**

Variable are a named location in memory where data can be stored.

**const fruit = "Watermelon"; // JavaScript**

**int id = 1002937845; // Java**

**$tools = array["Hammer", "Wrench", "Screwdriver"]; // PHP**

The process of carving out a piece of memory and assigning it a name is called creating, allocating or **declaring** a variable.

**const myName;**

**const phonebookArray;**

We have not given these variables values, we have just made them exist. In JavaScript we use the keyword **const**, **let** or **var** when first declaring a variable.

When we give a variable a value, or simply change its value to something else we are **assigning** that variable some piece of data. We no longer use the keywords because the variable has previously been created further up the code.

**myName = “John Murphy”;**

**phonebookArray = [ 123, 456, 789 ];**

**myName = “Joan Murphy”;**

The examples above are both assigning a new value to memory slots named '**myName**' and '**phonebookArray**'.

In strongly typed languages (Java, C++) we need to specify the data type of variables.

**string presidentName = "Joe Biden";**

**float circleArea = pi \* r \* r;**

In strongly typed languages, assigning the wrong data type to a variable will result in a syntax error.

**integer year = "Two Thousand and Twenty" --- ERROR**

JavaScript, Python and PHP are examples of weakly-typed languages and therefore you do not need to specify the variable/data type.

**Arrays**

We mentioned earlier that arrays are a special type of variable. They are a single variable which can hold more than one value at a time. Think of them as a collection, a list or a bag of items. Using arrays we can sort through items, access specific items using an index, filter items and loop through every element inside the array. Arrays can contain other variables including variables of a mixture of types and even other arrays! The term **item** or **element** is used to describe what is contained in an array.

In JavaScript an array:

* Can contain the same element more than once (can contain non unique items). JavaScript has something called **Sets** which only admit unique elements.
* Can change, add or remove elements,
* Has its elements ordered (indexed). The first item in it is in position 0. The third item is in position 2.

Python has 4 different types of collection variables that perform similarly to JavaScript arrays. They are called **lists**, **tuples**, **sets** and **dictionaries** and differ based on what was mentioned above.

**Creating Arrays in JavaScript**

**Wrong Way**

The second way is ill advised as it may cause confusion with other objects. I’m just showing you here because you may come across it when looking at old code.

**var myArray = new Array("red", "green", "blue", "yellow"); //** *Avoid this way*

**Correct Way**

The first way is the correct way which is what you should use. It basically just involves using square brackets **[]**

**var myArray = ["red", "green", "blue", "yellow"] ; //** This is good

**Types of Arrays in Programming Languages**

There are 3 main types of arrays:

* **Indexed Arrays:** are the ones shown above. The items stored in the array can be *accessed using a number*. The first element in the array is referred to as 0 while the next is 1 and so on. **myArray[0]** is equal to “red”.

Indexed arrays are the most common and the easiest to use when looping through big arrays.

**alert( myArray[1] ); //** *This alerts “green”*

**alert( myArray[3] ); //** *This alerts “yellow”*

* **Associative Arrays:** All items in the array are linked to a key which you would use to access them instead of referring to a number. These aren’t in JavaScript however they are present in PHP.

**<?php**

**$fruitPrice = array("Apple"=>"$0.65", "Banana"=>"$0.80", "Orange"=>"$0.75");**

**?>**

In this case when we want to get the value/price of the element known as “Apple” we must put “Apple” in the square brackets following the array name.

**$fruitPrice[ "Banana" ];**

* **Multidimensional Arrays:** These are arrays inside of arrays. Sort of like the movie “Inception”, when there are multiple arrays inside of arrays it can get quite confusing. The most common multi array you may come across is a 2 dimensional array. It is best to visualize them as a table first.

**Handy JavaScript Array Methods**

Like all programming languages there are a number of useful inbuilt methods and functions we can use to help us with arrays. Below are some important ones.

**.push(** *element to add* **)** -> Adding new element to end of array.

**myArray.push( "black" );**  **//** [“red”, “green”, “blue”, “yellow”, “black”]

**.length** -> Returns the number of elements in an array.

**myArray.length; //** returns 5

**.indexOf(** *element to check* **)** -> Checks to see if element is inside array. Returns number indicating the indexed position of the element. If element is not present in array then a number of -1 is returned. Remember 0 represents the first position in the array.

**myArray.indexOf( "blue" );**  **//** returns 2

There are many more useful methods to use with arrays. Removing, sorting, filtering, adding elements to specific locations, fusing elements into a single string to name but a few. It is very important that you become familiar with what they do. Check some of them out at the below links:

**W3Schools**: <https://www.w3schools.com/js/js_array_methods.asp>

**MDN**: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array> (check out left sidebar)

**Loops in JavaScript**

Repetition in programming is bad. By that we mean typing the same line time and time again is bad. Quite often you will find that many lines of code are repeated and frequently only slight parts between each line differ. This is especially true when dealing with arrays. Loops allow blocks of code to be repeated numerous times with the ability to change values with each increment.

If for example you wanted to create an alert for each number between 0 and 4 you could do it like this:

**alert(" the number 0");**

**alert(" the number 1");**

**alert(" the number 2");**

**alert(" the number 3");**

**alert(" the number 4");**

However a better way would be to use a **for loop**:

**for ( let i = 0; i < 5; i += 1 ) {**

**alert( "the number " + i );**

**}**

Here it is again with the specific sections highlights:

**for( let i = 0; i < 5; i += 1 ) {**

**//** *this block of code between the {} brackets gets repeated*

**alert( "the number " + i );**

**}**

A **for loop** is composed of 3 statements:

* **The initialisation** creates a variable and sets it as an integer. It ends with a semicolon **;**
  + An integer called **i** is created and is set to 0.
* **The condition** defines when the loop comes to an end. It ends with a semicolon **;**
  + The loop will end before **i** gets to equal 5. Therefore it will end after **i** equals 4.
* **The increment** increases (or decreases if you want) the initiated variable every time it gets to the end of the block of code. It **does not** end with a semicolon ;
  + After each time the block of code between the **{ }** brackets is run, **i** will increase by 1.

Any code within the curly brackets **{ }** gets run continuously for as many intervals that are set.

A **while loop** is slightly different than a **for** loop. Firstly, it is comprised of one statement, the **condition**, and the **initialisation** variable must be placed outside the code block altogether. The **incrementation** should occur inside the loop otherwise you may have created an infinite loop.

**let i = 0;**

**while ( i < 5 ) {  
  alert( "The number is " + i );  
    i += 1;  
}**

A **do** **while loop** is very similar to a **while** loop however it guarantees that the loop will run at least once. After this first iteration in the **do** block, the **condition** is checked and the loop continues so long as it evaluates to true.

**let i = 0;**

**do {**

**alert( "The number is " + i );**

**i++;**

**}**

**while ( i < 10 );**

Note if you forget to increase the variable used in the condition, these loops will never end…

*The following loops are used exclusively to iterate over collection data structures like arrays and objects.*

A **for of** loop is composed of a single statement and is used to iterate over arrays. The loop will continue until all items inside the array have been iterated over.

**const myArray = ["Canada", "USA", "Mexico"];**

**for ( let item of myArray ) { //** *item becomes the element in the array*

**alert( item );**

**}**

A **for in** loop is composed of a single statement and is used to iterate over objects. It can also be used on arrays however the order of the array may not be preserved so it is not advised. **for in** loops are best used on objects and not arrays. The loop will continue until all items inside the object/array have been iterated over.

**const myArray = ["Canada", "USA", "Mexico"];**

**for ( let itemPos in myArray ) { //** *item becomes the position or key of the item depending on whether an array or object is being iterated over*

**alert( myArray[ itemPos ] );**

**}**

**Looping Through Arrays**

Loops are used extensively to iterate through arrays in programming. Since arrays are indexed, it means that we can use an increasing number to run through each element in the array. Simply by adding the initialisation ( **i** ) variable inside the array index section **animalArray[ i ]** and having this in the for loop, we will have access to everything.

**const animalArray = [ "cat", "dog", "elephant" ];**

**for ( let i = 0; i < 3; i += 1 ) {**

**alert( animalArray[ i ] );**

**}**

The above code can be improved however. As you can see the for loop will **run 3 times** as **i** becomes 0 and then 1 and then 2 before the loop stops since **i** will no longer be less than 3. However if we added new elements to the array then we would have to update the **for** loop to account for this. Our code is not very robust.

By replacing the number 3 above in the conditional section of the code to **animalArray.length** it will ensure that the **for** loop will run only for as many times as there are elements inside the array. Therefore it is safe and practical. Remember **animalArray.length** is equal to 3 because it has “cat”, “dog” and “elephant” inside of it. Note that you are using **< animalArray.length** and not **<= animalArray.length** because the index position of arrays start at 0 and not 1. Therefore, the last element in the array is in position 2 which is less than the number of items in the array which is 3.

**for ( let i = 0; i < -animalArray.length-; i += 1 ) {**

**alert( animalArray[ i ] );**

**}**

**break; and continue; in Loops**

If you want to jump out of a loop you can use the **break;** keyword. When this is read by the interpreter the loop ends immediately.

If you want to skip a single iteration of a loop and move onto the next one you can use the **continue;** keyword.

**for ( let i = 1; i < 100; i += 1 ) {**

**if ( i >= 6 ) {**

**break; //** *Ends loop*

**}**

**if ( i % 2 === 0 ) {**

**continue; //** *Skips the rest of this iteration for any number divisible by 2*

**}**

**alert( i );**

**}**

The alerts would be 1, 3, 5