**2. Basics**

**Variables**:

In programming we use a variable to store data temporarily in a computer memory. We store our data somewhere and give that memory location a name for future use. In JavaScript we can declare variable by using three keyword var, let, and const.

**Variable declaration rules in JavaScript**:

1. Cannot be a reserved keyword (this, if, ...)
2. Should be meaningful
3. Cannot start with a number (1name)
4. Cannot contain a space or hyphen (-)
5. Are case-sensitive
6. Modern best practice is to declare each variable separately

let firstName = "Ruhul";

let lastName = "Amin";

**Difference between let vs const vs var in JavaScript**:

The difference is scoping. var is scoped to the nearest function block and let is scoped to the nearest enclosing block, which can be smaller than a function block. Both are global if outside any block.

**Global**:

They are very similar when used like this outside a function block.

let me = "go"; *// globally scoped*

var i = "able"; *// globally*

`const` is similar to C or C++'s `const` only. Once we assign a value to the variable declared as `const`, we cannot assign some other value to the const variable.

However, global variables defined with let will not be added as properties on the global window object like those defined with var.

console.log(window.me); *// undefined*

console.log(window.i); *// 'able'*

**Function**:

They are identical when used like this in a function block.

function ingWithinEstablishedParameters() {

let terOfRecommendation = "awesome worker!"; *//function block scoped*

var sityCheerleading = "go!"; *//function block scoped*

}

**Block**:

Here is the difference. let is only visible in the for() loop and var is visible to the whole function.

function allyIlliterate() {

*//tuce is \*not\* visible out here*

for (let tuce = 0; tuce < 5; tuce++) {

*//tuce is only visible in here (and in the for() parentheses)*

*//and there is a separate tuce variable for each iteration of the loop*

}

*//tuce is \*not\* visible out here*

}

function byE40() {

*//nish \*is\* visible out here*

for (var nish = 0; nish < 5; nish++) {

*//nish is visible to the whole function*

}

*//nish \*is\* visible out here*

}

**Redeclaration**:

Assuming strict mode, var will let you re-declare the same variable in the same scope. On the other hand, let will not:

let me = 'foo';

let me = 'bar'; *// SyntaxError: Identifier 'me' has already been declared*

var me = 'foo';

var me = 'bar'; *// No problem, `me` is replaced.*

Don't confuse `const` with mutable objects. As I said, you cannot assign some other value to the const variables, but you can modify the `const` value. For example,

const a = {};

a.a = 1;

is perfectly valid, because we are not changing what is assigned to `a`, but we are changing the object pointed by `a`

**Datatype in JavaScript**:

Programming languages all have built-in data structures, but these often differ from one language to another. Datatype can be used to build other data structures.

JavaScript is a loosely typed or a dynamic language. Variables in JavaScript are not directly associated with any particular value type, and any variable can be assigned (and re-assigned) values of all types:

We have two types of programming languages

a) static

b) dynamic

In static languages when we declare a variable the type of the variable is set and we cannot change the type in future.

In a dynamic language like JavaScript we can change the type of a variable after declaration.

var foo = 42; *// foo is now a number*

foo = 'bar'; *// foo is now a string*

foo = true; *// foo is now a boolean*

The latest ECMAScript standard defines seven data types:

* Six data types that are primitives:
  1. Boolean
  2. Null
  3. Undefined
  4. Number
  5. String
  6. Symbol (new in ECMAScript 6)
* And Reference type
  1. Object
  2. Array
  3. Function

**Primitive or value type**:

A primitive (primitive value, primitive data type) is data that is not an object and has no methods. In JavaScript, there are 6 primitive data types: string, number, boolean, null, undefined, symbol (new in ECMAScript 2015).

Most of the time, a primitive value is represented directly at the lowest level of the language implementation.

All primitives are immutable, i.e., they cannot be altered. It is important not to confuse a primitive itself with a variable assigned a primitive value. The variable may be reassigned a new value, but the existing value cannot be changed in the ways that objects, arrays, and functions can be altered.

**Example**:

*// Using a string method doesn't mutate the string*

var bar = "baz";

console.log(bar); *// baz*

bar.toUpperCase();

console.log(bar); *// baz*

*// Using an array method mutates the array*

var foo = [];

console.log(foo); *// []*

foo.push("plugh");

console.log(foo); *// ["plugh"]*

*// Assignment gives the primitive a new (not a mutated) value*

bar = bar.toUpperCase(); *// BAZ*

**Example**:

how JavaScript deal with Primitives.

*// The Primitive*

let foo = 5;

*// Defining a function that should change the Primitive value*

function addTwo(num) {

num += 2;

}

*// Another function trying to do the same thing*

function addTwo\_v2(foo) {

foo += 2;

}

*// Calling our first function while passing our Primitive as an argument*

addTwo(foo);

*// Getting the current Primitive value*

console.log(foo); *// 5*

*// Trying again with our second function...*

addTwo\_v2(foo);

console.log(foo); *// 5*

In primitive type we have

*//string*

let name = "Ruhul";

*//number*

let number = 10;

*//boolean*

let boolValue = false;

*//undefined (undefined is a type and a value)*

let myNumber;

*//null*

let selectedColler = null;

**Reference type**:

Variables that are assigned a non-primitive value are given a reference to that value. That reference points to the object’s location in memory. The variables don’t actually contain the value.

In Reference type we have

1. object
2. array
3. function

**Object**:

JavaScript is designed on a simple object-based paradigm. An object is a collection of properties, and a property is an association between a name (or key) and a value. A property's value can be a function, in which case the property is known as a method. In addition to objects that are predefined in the browser, you can define your own objects.

**Example**:

let person = {

name: "ruhul",

age: 30,

job: "programmer"

};

here person is an object.

There are two ways to change the property of an object.

*//Dot notation*

person.name = 'John';

*//Bracket notation*

person['name'] = 'Reza';

**Arrays**:

Array is a data structure that use to represent a list of data. In JavaScript array is heterogeneous and dynamically we can change array size in JavaScript;

let selectedColores = [];

here selectedColler is an empty array and [] is array literal.

**Functions**:

Functions are one of the fundamental building blogs in JavaScript. A function is a set of statement that perform a task or calculate a value.

**Example**:

function greet(name, lastName) {

console.log("Hello " + name + " " + lastName);

}

greet("Ruhul", "Amin"); *//Hello Ruhul Amin*

greet("John", "Smith"); *//Hello John Smith*

**Types of functions**:

*//perform a task*

function greet(name, lastName) {

console.log("Hello " + name + " " + lastName);

}

greet("Ruhul", "Amin"); *//Hello Ruhul Amin*

greet("John", "Smith"); *//Hello John Smith*

*//calculate a value*

function square(number){

return number\*number;

}

let number = square(10);

console.log(number); *//100*

As a function argument we can pass a value or a expression.

2. Basics