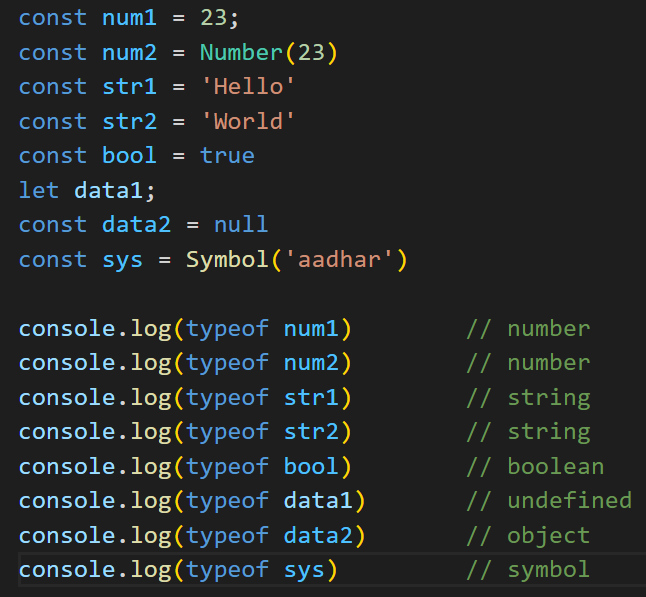
**DATA-TYPES: PRIMITIVE and NON-PRIMITIVE**

**Dynamically typed**

* JavaScript is dynamically typed; you do not need to declare a variable’s data type before using it. Instead, based on the value assigned to the variable, JavaScript will automatically determine the data type.

**Data types in JavaScript**

1. ***Primitive (7)*** – Number, String, Boolean, BigInt, Undefined, Null\* and Symbol
2. ***Non-primitive or Reference Types*** - Objects(Arrays, Functions)  
     
   **Primitive data types**

* predefined or built-in data types provided by JavaScript
* immutable (values cannot be changed once they are assigned)
* primitive-type variable “contains” primitive data
* Primitive data types are stored directly in memory
* When it comes to primitive data types, their values are what are compared. Two variables with the same primitive value are considered equal.  
  

**Primitive – Number**

* includes both integers and floating-point numbers
* typeof returns number  
   A number of numbers and symbols

  Description automatically generated with medium confidence

**Primitive – String**

* represents sequences of characters enclosed in single or double quotes.
* more details (in other document)
* typeof returns string  
   A black and white image of a symbol

  Description automatically generated

**Primitive – Boolean**

* used for conditional statements and logical operations
* has only two possible values: true and false
* typeof returns boolean  
   A black text with black letters

  Description automatically generated

**Primitive – Undefined**

* represents a variable that has been declared but has not been assigned a value yet
* typeof returns boolean  
   A black and white text

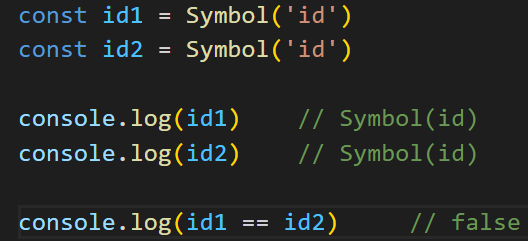
  Description automatically generated

**Primitive – Null**

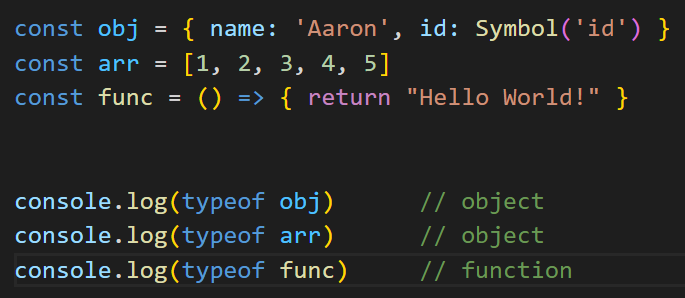
* used to indicate the absence of a value or a deliberate non-value
* typeof returns object (interesting)  
   A black and white text

  Description automatically generated

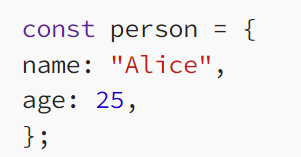
**Primitive – Symbol**

* *Symbol(keyname)*
* typeof returns symbol
* represents a unique and immutable value, often used as object property keys
* say for example , you have multiple objects, and we assign each of them an id key.  
  Now we don’t have a particular id for each, but we want them to be unique. That’s where we use Symbols.  
  

**Non-Primitive data types**

* mutable (values cannot can be changed after they are assigned)
* referenced data types
* All non-primitive types are in a way an object
* non-primitive data types are stored as references to their values in memory.
* non-primitive type variable “reference” to non-primitive data
* printing out a variable that references a non-primitive data, fetches us not the address (like C programming), but the non-primitive data
* in JavaScript, a reference variable has direct access to non-primitive data it is referencing to (no need to use pointers like C). That’s how JavaScript works
* JavaScript simplifies the work of typeof by returning 'object' whenever the given expression is a reference to some data in memory.
* When comparing non-primitive data types, their references are compared. Two variables that reference the same object are considered equal.  
    
   

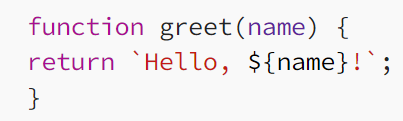
**Non-Primitive – Object**

* A versatile container that can hold key-value pairs and eclosed in {} braces
* Objects can store functions and other objects, making them a fundamental part of JavaScript.
* Object is a value in memory which is possibly referenced by an identifier
* Object properties can be accessed via:  
   - objectName.propertyName  
   - objectName["propertyName"]
* An object is a mutable data structure
* typeof returns -> object  
   

**Non-Primitive – Array**

* specialized form of an object used to store ordered lists of values
* Arrays can store elements of different data types and are accessed by numerical indices.
* Arrays fall in the “object” category of JavaScript data types because they are stored by references and NOT by their actual values.
* An array is indeed a reference to an ordered bunch of data in memory and likewise translates to 'object' when inspected by the typeof operator.
* typeof returns object ****

**Non-Primitive – Function**

* Functions are also objects with the additional capability of being callable.
* They can be assigned to variables, passed as arguments, or returned from other functions.
* Functions fall in the “object” category of JavaScript data types because they are stored by references and NOT by their actual values.
* typeof returns function  
   ****

**Non-Primitive –** **Custom Objects**

* Developers can create custom non-primitive data types by defining their own object structures.
* This allows for more specialized data storage and manipulation.

**PRIMITIVE vs NON-PRIMITIVE DATA TYPES (difference)**

1. ***Mutability***: Primitive data types are immutable, meaning their values cannot be changed once assigned. Non-primitive data types are mutable and can be modified.
2. ***Storage***: Primitive data types are stored directly in memory, whereas non-primitive data types are stored as references to their values in memory.
3. ***Comparison***: Primitive data types are compared by value, while non-primitive data types are compared by reference.

*IN ORDER TO UNDERSTAND data types, we should have a clear understanding of:*

a. primitive value  
 b. non-primitive value  
  
**Assign AND Copy variable – PRIMITIVE TYPES**

***Primitive data-types (7):*** String, Number, BigInt, Symbol, Boolean,Undefined, Null

***Primitive values:*** “hello”, 3, Symbol(“id”),true, null etc etc

* Primitive values are data that are directly stored in a variable
* a primitive-type variable actually contains “primitive data” and not any reference

**Assign variable a primitive value:**

* When we assign a primitive value to a variable, a copy of that value is created and stored in memory. Any changes made to the variable do not affect the original value.  
   *Process:*
* primitive-data with value of 23 created (in memory)
* copy of “data with value 23” created (in memory) happens AND is stored into x
* x contains actual value 23 ( and not reference )  
    
  A black rectangles with black text and a black arrow

  Description automatically generated

**Copy of a primitive variable:**

* Copying a primitive variable copies its “value” and stores in new variable.
* The new variable now has its own copy of value  
   *Process*:
* primitive-data with value of 23 created (in memory)
* copy of “data with value 23” created (in memory) AND is stored in x
* x contains actual value 23 ( and not reference )
* copy of “data contained in x” happens AND is stored in y
* y contains actual value 23 ( and not reference )  
    
  A diagram of a graph

  Description automatically generated
* In the example above, the variable y is assigned a copy of the value of x.
* When we change the value of y, it does not affect the value of x. This is because x and y are separate variables with separate memory locations.

**Assign AND Copy variable – NON-PRIMITIVE TYPES**

***Non-primitive data-types* :** Array, Object, Function etc

***Non-primitive values* :** {name: ‘Nishant’}, [1,2,3,4] function(){…}

* a non-primitive-type variable contains only a “reference to data” and does not contain the data itself.
* Reference or non-primitive values, on the other hand, are objects that are stored in memory and accessed through a reference.

**Assign variable a non-primitive value:**

* When we assign a reference(or non-primitive) value to a variable, a reference to the original value is created and stored in memory. Any changes made to the variable affect the original value.
* In JavaScript, objects are assigned and copied by reference.
* This means that a variable stores not the "object value", but a "reference" (address in memory) for the value
* When you assign a variable to an object, you are not creating a copy of the object (like primitives). You are simply creating a new reference to the object.
* We can access the object's properties and methods directly through the reference (or variable). We don’t get address of the object via variable (because I don’t know and because that’s not how js works)  
   *Process:*
* non-primitive-data with value {name: “Ankit”, age: 18} created in memory
* variable “obj” starts pointing to above created object, thereby acting as a reference to the object.
* Note that obj does not contain object data in itself
* However, to access object data, we can simple use obj (it doesn’t return address of obj, that’s how js works)
* use obj.name and obj.age to access object key-value pairs
* if we do, obj.name = “Kunal”, it will modify main object data itself, thereby meaning, objects or non-primitives are “mutable”  
  A black and white image of a memory card

  Description automatically generated

**Copy of a non-primitive value:**

* Copying a reference variable copies that reference, not the object
* If we assign a reference variable to another variable, both variables will point to the same object. (meaning both are reference variables)  
   *Process:*
* non-primitive-data with value {name: “Ankit”, age: 18} created in memory
* variable “obj” starts pointing to above created object, thereby acting as a reference to the object.
* const obj2 = obj
* obj2 copies the reference of obj only, thereby pointing to obj data
* hence obj2 and obj both are identical and referencing the same object
* since both reference same object, both are equal ===
* Note that if: obj2.name = “Kunal”, it will update original obj data.
* Hence now both obj2 (and obj) will have name as “Kunal”
* So be very careful when working with non-primitives as one change is a *change-for-all*  
    
  A black and white diagram of a memory

  Description automatically generated

**Primitives and functions**

* When we pass a primitive value as an argument to a function, a copy of that value is created and passed to the function.
* Any changes made to the variable inside the function do not affect the original value.  
  A computer screen with text and numbers

  Description automatically generated

**Non-primitives and functions**

* When we pass a reference value (or non-primitive-type variable) as an argument to a function, a reference to the original value is passed.
* Any changes made to the variable inside the function affect the original value.  
    
  A computer screen with text and numbers

  Description automatically generated with medium confidence