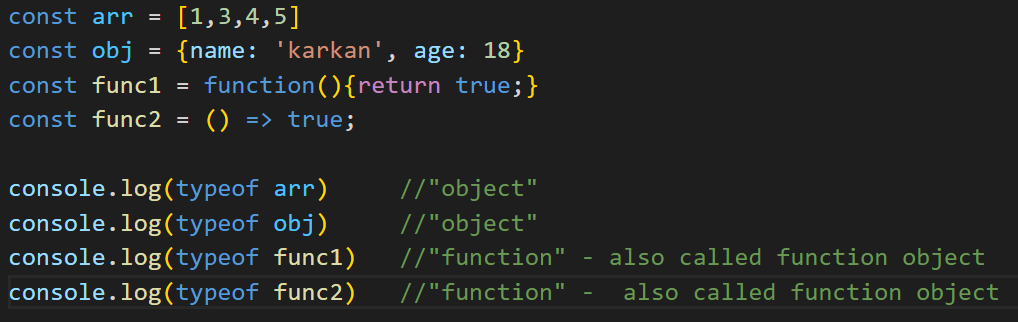
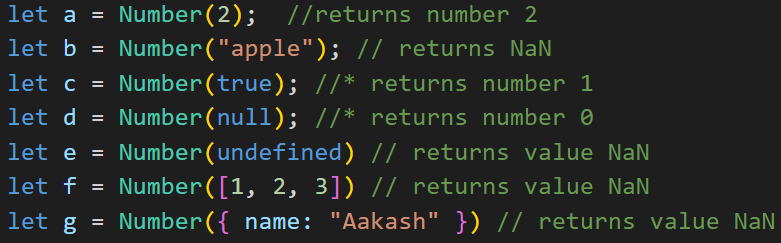
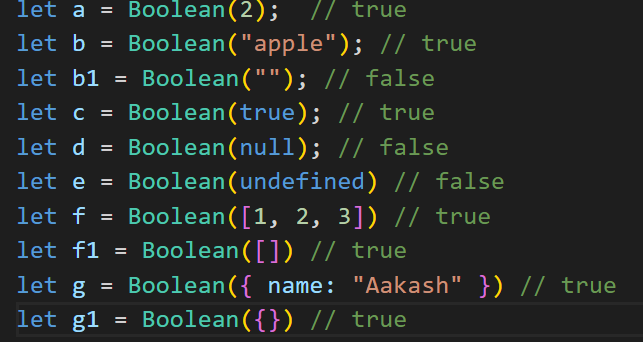
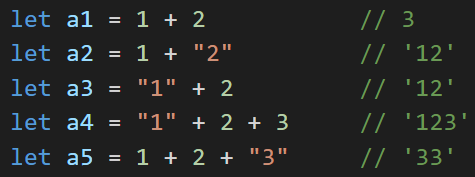
//JavaScript

1. **Compilers**:  
   - Translate the entire source code into machine code before the program runs. This results in faster execution since no translation is needed during runtime.
2. **Interpreters**:  
   - Translate code written in a high-level programming language into machine code line-by-line as the code runs.  
   - Interpreters translate one statement at a time, create an exe of the programming language, and execute source statements line by line.  
   - This makes its overall speed and execution slower than a compiler.
3. Don’t assign a variable as “name” outside of functions i.e. globally. In browser, global variable “name” has a special meaning. However, if you assign it inside a function, no issues. Go ahead!
4. **console.table:**  
   - console.table([var1,var2,var3])  
   - Just like console.log, it displays content in a table.  
   - var1 comes at 0th index of table, var2 at 1st and so on.  
    A black and yellow text

   Description automatically generated with medium confidence
5. **const variable:**  
   - value once provided cannot be changed  
   - value to be provided at the time of initialization only.  
   - const id = “1234”
6. **let variable:**  
   - values and data type can be changed later  
   - only declaration is sufficient, value can be provided later.  
   - let username;  
    username = “Ankit”  
    username = [1,2,3]
7. **Primitive data type**:  
   - number - null -BigInt  
   - string - undefined  
   - boolean - Symbol
8. **Non-primitive data type:**  
   - Array  
   - Object  
   - Function
9. **typeof – non primitives:**  
   - all are in one way or the other of object type(s)  
   - a variable of data-type as “function” is also called as function object  
   - objects and arrays are of data-type as “object”  
   
10. **null:**   
    - It means the variable is empty  
    - Doesn’t mean 0, doesn’t mean empty string, it just means the variable is empty, there’s nothing there.
11. **typeof operator:**- used to check data-type of a variable  
    - typeof 2 *//Number*  
     typeof “apple” *//string*  
     typeof true *//boolean*  
     typeof null *// object\* //and not null, that’s how js is written*  
     typeof undefined *//undefined*
12. **Data type conversions:**  
    - In JavaScript, many times a need arises where we need to convert one data type to another. Sometimes it is feasible, while many a times it is not.
13. **Number() type conversion:****- In above example we can see type conversion from a [string/boolean/null/undefined,array,jsonobject] to number   
    - Many times it’s not possible to convert a given data type to a number, hence it returns NaN, meaning Not a Number.  
    - Surprising thing about NaN is when you check its type:  
     - console.log(typeof NaN) //returns a number (LOL)
14. **Boolean type conversion:**  
      
    - In above example, conversion of [string/number/null/undefined,array,jsonobject] to boolean
15. **String conversion:**  
      
    - Convert any type to a string, it will add quotes around it  
    - [number/boolean/null/undefined,array,jsonobject] to string
16. **Auto conversion:  
    -** When performing certain operations, JavaScript converts one data type to another based on certain rules setup by ECMAScript directives.  
      
    - In the above snippet, we can see numbers being converted to string. Observe the difference b/w a4 and a5
17. **Increment ++ operator (Prefix and Postfix):**  
    - it means increment the value by 1  
    - all concepts similar for -- decrement operator (means decrease value by 1)  
    ***Postfix***:   
     - first returns the “immediate” value, then increments the value  
     - let i=0;  
     console.log(i++) // *prints 0*  
     console.log(i) // *prints 1*  
    - let y = 0;  
     let n = y++ // first assign the immediate value of y to n, then increment value of y  
     console.log(n) //*prints 0*  
     console.log(y) //*prints 1*  
    ***Prefix***:  
     - first increment the value, then return the incremented value   
     - let i=0;  
     console.log(++i) // *prints 1*  
    - let y = 0;  
     let n = ++y // first increment value of y, then assign the incremented value of y to n  
     console.log(n) //*prints 1*  
     console.log(y) //*prints 1*
18. **Comparison operators:**== *equal to (data check)*

!= *not equal to (data check)*

=== *strictly equal to (data and data-type check)*

!== *strictly not equal to (data and data-type check)*

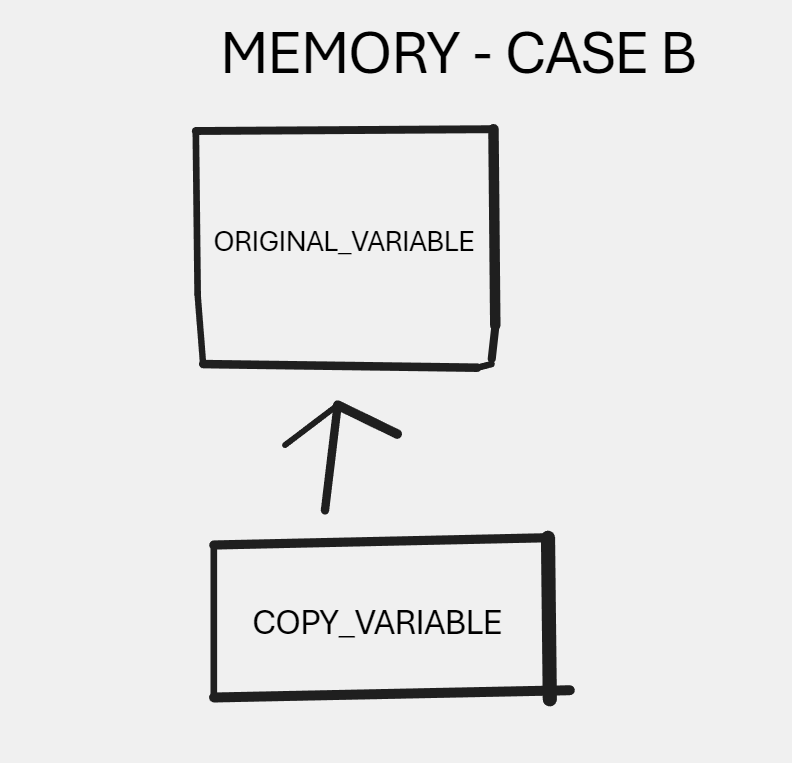
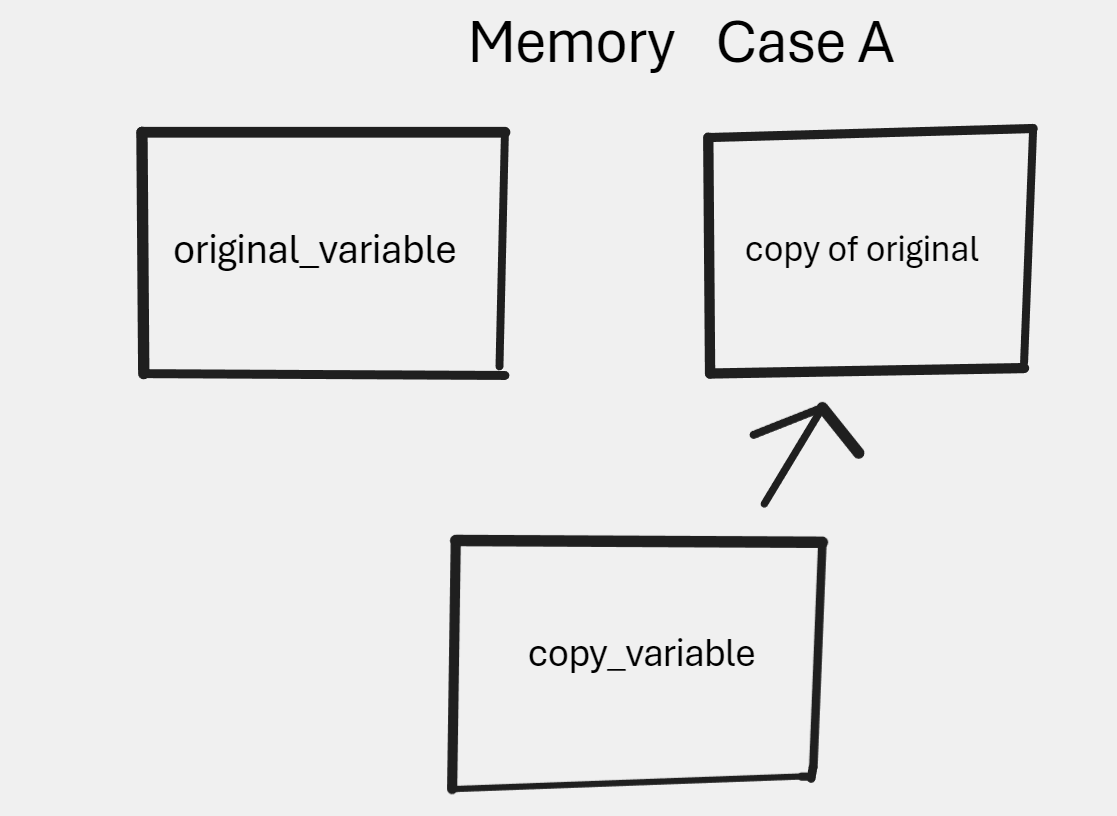
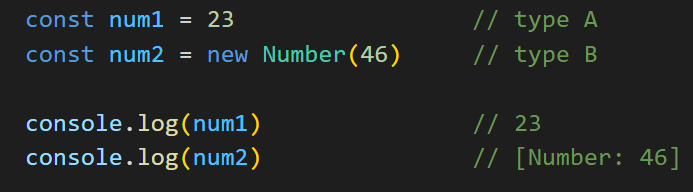
< *less than (data check)*

> *greater than (data check)*

<= *less than or equal to (data check)*

>= *greater than or equal to (data check)*

1. **Comparison operators – auto convertion:**  
   - when using above operators, they cause data type conversion as in Index 13  
   A screenshot of a computer

   Description automatically generated  
   - for checks related to null and undefined, see code
2. **Copying in JavaScript:**  
   Consider a scenario,  
    let a;  
    a = //some value assigned  
    b = a;  
    a is called as original\_variable and b is called copy\_variable  
    “copy of original” means initially it has same value of original\_variable but at a different memory location.  
   - When trying to copy a variable of a certain data-type in JS, two possibility arises:  
    ***Case A.*** a new copy of original\_variable created, and the copy\_variable references the copy of original variable  
    ***Case B.*** copy\_variable references directly to the original\_variable  
   - In case A, when modifying the copy\_variable, the original\_variable remains untouched  
   - In case B, if copy\_variable is modified, original\_variable is modified as well.  
      
     
   - When copying primitive data types, a copy of original\_variable is created and copy\_variable points to it. (Case A)  
   - For non-primitive data-types, cases vary.  
   - When copying an array of numbers – Case A  
   - When copying named functions, Case A  
   - When copying objects – Case B, hence original\_object gets modified  
   - When copying array of objects, Case B  
   - See code 12\_memory-non-primitive.js for references
3. **Template strings/String interpolation:**
4. **Numbers in JavaScript:  
   -** Two ways to declare a number in JavaScript  
     
   - typeB returns a number object and type A returns primitive number  
   - typeB has some methods already attached to it (ex-toString) while typeA has no methods attached.  
   - However all methods that type B has access to can be applied to type A too (JavaScript does that behind the scenes)  
   - Overall, no major difference b/w the two and we can use anyone.
5. **Number Method – toString()**  
   - converts a number to a string  
   - returns a string  
   - Number.toString()  
   A screen shot of a computer

   Description automatically generated
6. **Number Method – toFixed()  
   -** used to limit decimal digits of a number  
   - returns a number  
   - Number.toFixed(limit)  
   A screen shot of a computer

   Description automatically generated
7. **Number Method – toLocaleString()**  
   - used to convert numbers into more "readable" format as per locale country defined ( adds commas as per lakhs or millions etc)  
   - returns a string  
   - Number.toLocaleString(locale)  
   - locale examples – Indian (en-IN), USA (en-US) etc  
     
   A screenshot of a computer program

   Description automatically generated
8. **Math function**:  
   A. Math.round - round off to closest whole number

B. Math.ceil - round off to higher whole number

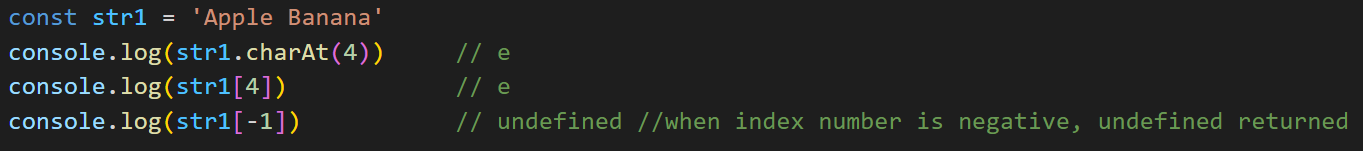
C. Math.floor - round off to lower whole number

D. Math.ceil - round off to higher whole number

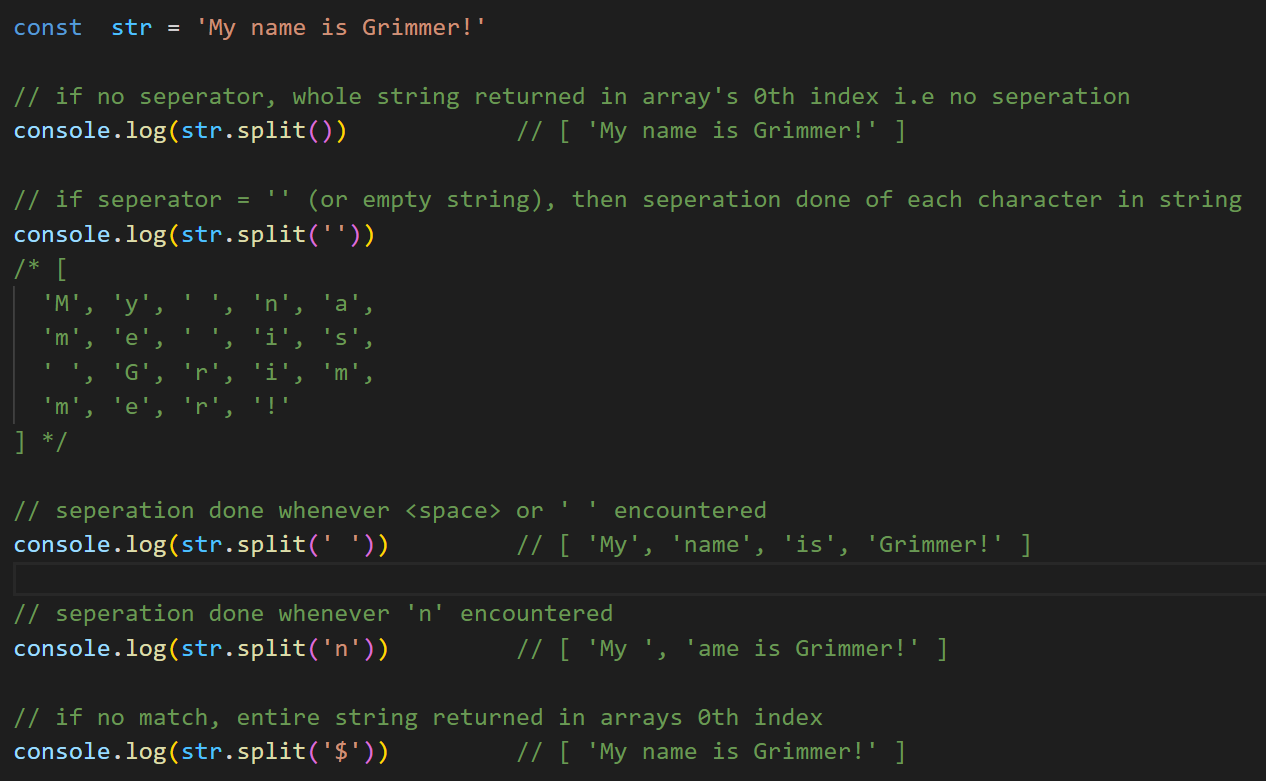
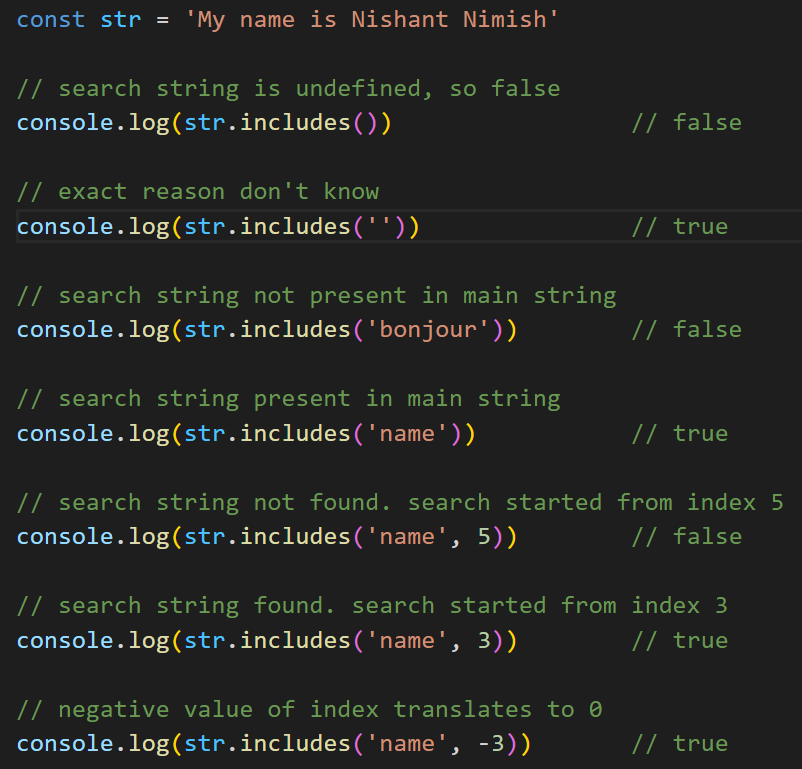
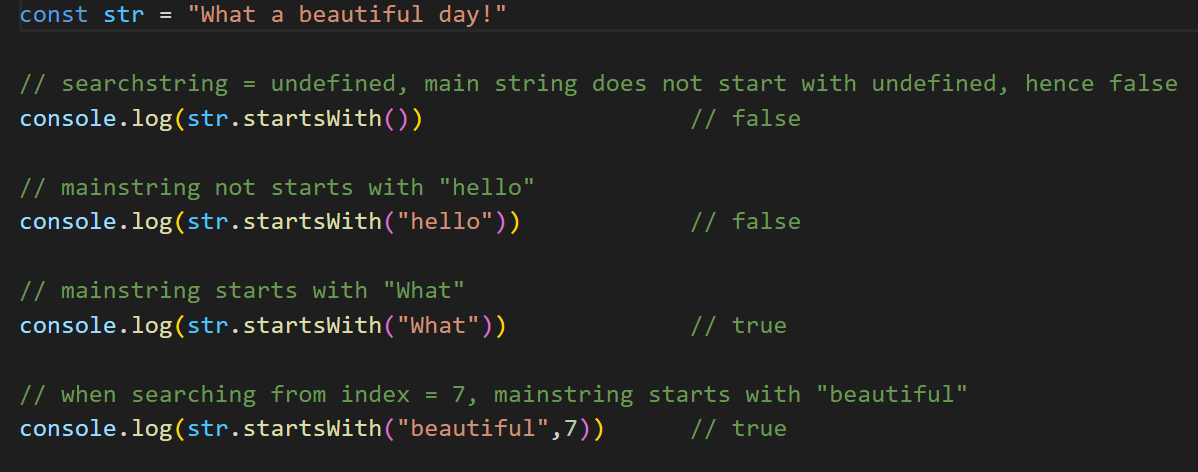
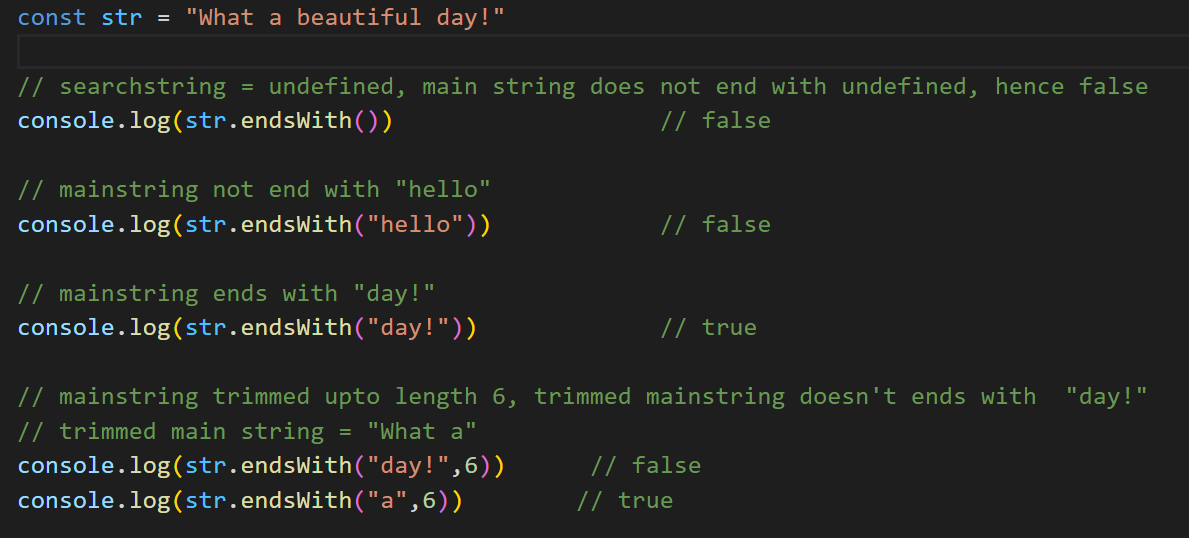
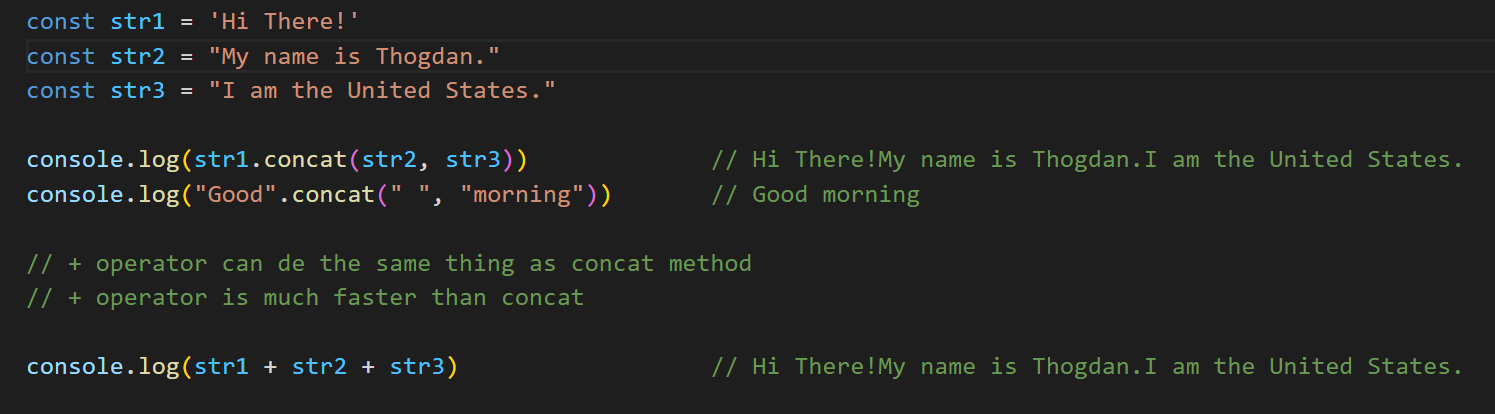
E. Math.random - every time returns a random number b/w [0,1]  
A black background with text and numbers

Description automatically generated

1. String
2. **String property – length**  
   - one and only property of string (rest are methods)  
   - returns a number (calculates number of characters in string)  
   A computer screen shot of a computer code

   Description automatically generated
3. **String Method – charAt()**  
   - String.charAt(index)  
   - used to access a certain character at a certain index in a string  
   - returns a character value at specified index  
   - returns undefined if index is negative  
   - *Alternative way to access a character – str[index]*  
   
4. **String Method - indexOf and lastIndexOf**  
   - mainString. indexOf(searchstring, startindex:*optional*)  
   - mainString. lastIndexOf (searchstring, startindex:*optional*)  
   - both methods perform a search operation over the string  
   - *indexOf*: parses the string from first index to last (index 0 to length-1)  
   - *lastIndexOf*: parses the string from last index to first (length -1 to 0)  
   - if searchstring pattern present in mainstring, index of “first” such occurrence is returned  
   - search operation is case-sensitive  
   - if search string not present, -1 returned  
   - if startindex is negative, it is auto-converted to 0  
   - mainstring.indexOf(searchstring,5): search operation will start from index = 5 to (mainstring.length -1) (L to R)  
   - mainstring.lastIndexOf(searchstring,5): search operation will start from index = 5 to 0 (R to L)  
   A screen shot of a computer program

   Description automatically generated
5. **String Method – slice**- String.slice(startindex, stopindex:*optional*)  
   - returns a “part” of a string  
   - string gets trimmed from index: [startindex,stopindex)  
   **-** if stopindex not defined, by default it is = string.length -1 **-** if startindex = stopindex, empty string returned **-** if startindex > stopindex, empty string returned **A screen shot of a computer program

   Description automatically generated**  
   - startindex and stopindex can be negative (for more details refer code)  
   - Note: negative values of indexes work differently with slice ( -1 represents last index etc)
6. **String Method – substring**- Same as slice with below major difference:  
   - if start > stop, indexes are swapped behind the scenes  
   - any negative value of index translates to 0
7. **String Method – split**- String.split(separator)  
   - returns an array of strings  
   - the string is parsed and each time the separator appears in the string, left substring of separator is stored as new entry in array  
   - In simple words, an individual 'string' gets split into an 'array of strings' based on seperator defined  
   - if no seperator, whole string returned in array's 0th index i.e. no separation  
   - if no seperator, whole string returned in array's 0th index i.e. no separation  
   - if seperator = '' (or empty string), then separation done of each character in string (See screenshot)  
   ****
8. **String Method – includes**- mainstring.includes(searchstring, startindex:*optional*)  
   - returns Boolean  
   - if searchstring present inside mainstring, return true, else false  
   - at least one occurrence is needed for result to be true  
   - negative value of startindex translates to 0 ****
9. **String Method – startsWith  
   -** mainstring.startsWith(searchstring, startindex:*optional*)  
   - returns a Boolean  
   - verifies if the mainstring “starts with” searchstring  
   - verification will start from startindex, if not defined, by default = 0  
   ****
10. **String Method – endsWith  
    -** mainstring.endsWith(searchstring, length:*optional*)  
    - returns a Boolean  
    - verifies if the mainstring “ends with” searchstring  
    - if length defined, means mainstring trimmed to specified length and search operation performed on trimmed string  
    - by default, if not defined, length = mainstring.length  
    ****
11. **String Method – concat**- mainstring.concat(string1,string2...)  
    - concatenate multiple strings into one  
    - returns a new string  
    - ***“+” operator is also used to concatenate strings. It is much faster than concat method*  
    **
12. **String Method – repeat**- String.repeat(count)  
    - used to repeat a string ‘count’ number of times  
    - returns a string **A screenshot of a computer

    Description automatically generated**
13. **String Method – trim,trimStart,trimEnd**(a) *String.trimStart()* - trim whitespace at the start of string

(b) *String.trimEnd()* - trim whitespace at the end of string

(c) *String.trim()* - trim whitespace at start and end of string  
- whitespaces are trimmed at start and end of string BUT not in between  
- returns a string **A screen shot of a computer

Description automatically generated**

1. **String Method – padStart,padEnd**- check code for details
2. **String Method – localeCompare**- mainString.localeCompare(compareString,locales:*optional*)  
   - locales: different languages have different set of alphabets. By default, its 'en'  
   - localeCompare checks the order in which characters appear in the language

- for example - 'a' letter comes before 'b' letter in English alphabet

- if mainstring comes before comparestring, a negative value is returned (different for different browsers)

- if mainstring comes after comparestring, a positive value is returned (different for different browsers)  
- if mainstring and comparestring have same order, 0 is returned  
  
**A screen shot of a computer program

Description automatically generated**

1. **String Method – search**
2. **String Method - match**
3. **String Method – matchAll**
4. **String Method – replace**
5. **String Method – replaceAll**