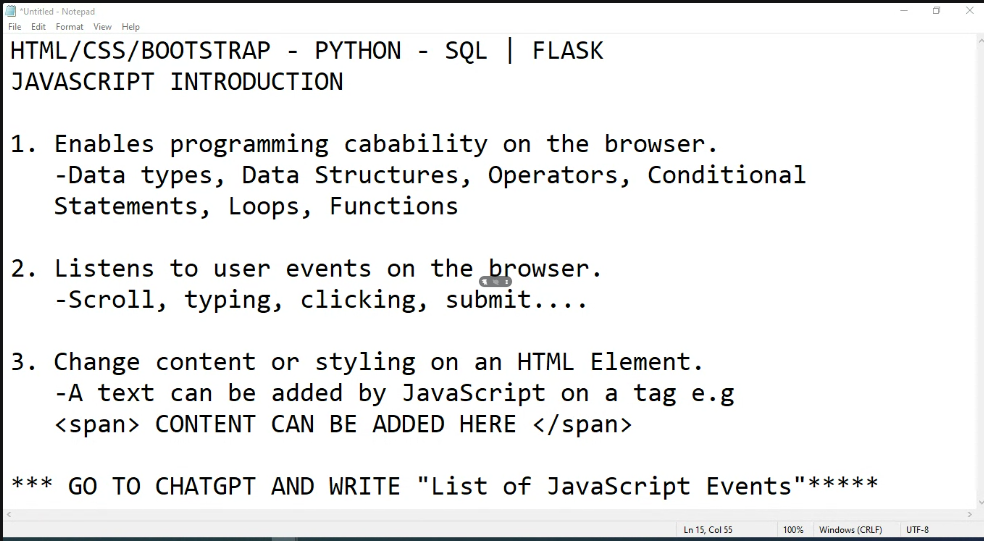
**task Html/css/bootstrap –Python – sql | flask**

**Javascript introduction**



UI- User Interface (HTML/CSS)

UX- User Experience (JavaScript)

FrontEnd Developer – HTML/CSS/JAVASCRIPT – Vanilla JavaScript (raw javascript)

**Using javascript**

1. Javascript is written at the bottom always.

Defining a variable

// console.log is print

        console.log('Hello world');

        // creating a variable

        let num1 = 2;

        let num2 = 5;

        let sum =num1 + num2;

        console.log(sum)

1.introduces programming on the browser.

Data types-strings \*\*methods\*\*\*\*

,numbers(ints,floats) \*\*\*arithmentic operations\*\*

, Booleans \*\*\* conditional operator\*\*\*logical operators\*\*

-Data structures –array(list) \*\*methods\*\*

- object (exactly like a dictionary)

--Operators –arithmetic - +, -

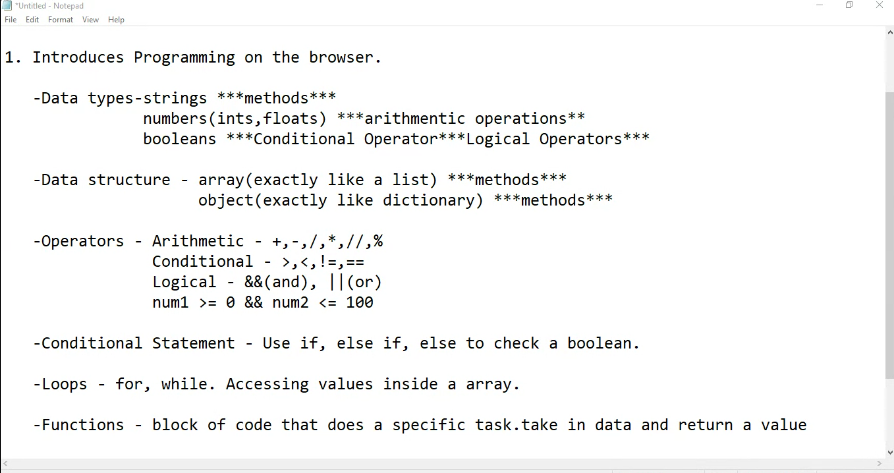
-conditional

-logical – and(&&), or (||) eg num1 >= 0 && num2 <= 100

-conditional statement – use if, else if, else to check a Boolean.

- loops – for for,while,. Accessing values inside an array.

- functions –block of code that does a specific task,take in some data and return a value.



**JS ARRAYS**

1. An array is a special variable/data structure used to hold more than one value at a point – equivalent to a list in python.
2. Properties
   1. They’re ordered – elements can be accessed with index.
   2. Mutable – can change / alter
   3. Can store multiple items of different data types

Syntax – let array\_name = [ values ]

There is one dimensional and multidimentional arrays

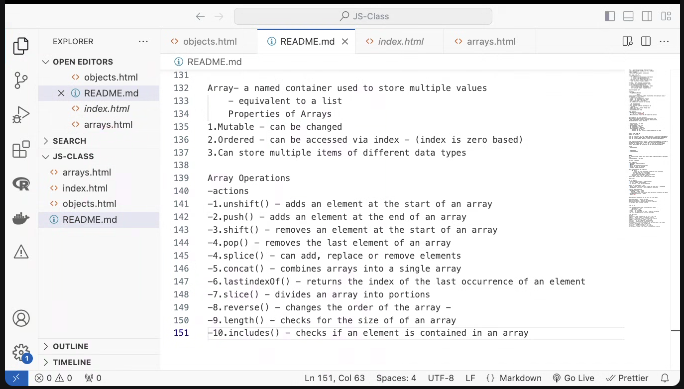
**Array operations**

Can range from creating arrays, updating values, adding and removing values

Task –slide 53, 69, 70

Use splice to add and replace elements

Use sort



**Objects**

Are data structures used to store values/key-value pairs. Equivalent to a dictionary in python

Key-value 🡪 property

1. No spaces
2. Can contain letters,digit, underscores and $
3. Can only start with\_ or letter or $

Xtics of objects

1. Store data in key value pairs.
2. Mutable
3. Object keys are essentially strings.
4. Values can be of any data type.
5. They have methods/operations.

Why use objects?

1. Can group related in a single container
2. Store complex structured data
3. To represent /model real world entities

Objects can be created in 2 ways:

1. Object literal syntax – directly creates key value pairs
2. Using object construct – built-in oject()

Accessing object properties

1. Using dot notation.
2. Using bracket notation.

TYPE CASTING

1.number

There is a difference between Uncaught TypeError and undefined

Uncaught TypeError cannot read properties..- you got to undefined point/ wrong key and still went further to try and access other nested objects. E.g

Undefined- you tried accessing a value using a wrong/undefined key – you stopped there.

**Optional chaining**

Uncaught TypeError cannot read properties..- you got to undefined point/ wrong key and still went further to try and access other nested objects. – this error will cause the code /program crash.

Optional chaining is a feature that allows you to safely access deeply nested properties of an object without worrying if part of the chain is null or undefined. – instead of throwing an error when you try to access a null/undefined property optional chaining returns undefined immediately and stops. Uses this symbol (?)

1. Once optional chaining encounters the first instance of null/undefined it stops at that point

**Spread operator**

1. Is feature that allows one to spread/expand /unpack elements of an iterable. Eg array,object,strings

Used to copy an array .(…)  let num = [1,2,3,4,5]

        let num2 =[...num]

// merging arrays

        let n = [1,2,3,4,5]

        let nu = [6,7,8,9,10]

        let num3 = [...n,...nu]

        console.log(num3)

 //coppying objects

        let table ={

            height:'90cm',

            width: '100cm'

        }

        let table2=[

            ...table

        ]

        console.log(table2)

        // merging objects

        let tabl ={

            height:'90cm',

            width: '100cm'

        }

        let tab ={

            height:'90cm',

            width: '100cm'

        }

        let table3={

            ...tabl,...tab

        }

        console.log(table3)

**operators**

1. Arithmetic. Eg. + ,-, \*,/,%,\*\*,++(incrementing),--
2. Comparison. Eg. >,<,==,!=, ===(strict equality operator-checks for the value and the type), !== strict inequality –value and type also
3. Assignment. Eg =, +=, -=,/=
4. Logical

AND – returns true if all are true -&&

OR – returns true if atleast one is true -||

NOT- returns the opposite value –boolean - !

The + is not unique operator, offers the operational overloading. The

**Strict equality operator**

Javascript has type conversion

// comparison operator

        // strict equality and inequality

        // -comparison in both value and data type. if == is used javascript will do type conversion.

        // - to return true operands have to be equal in both value and   data type

        let h = 5

        let j = '5'

        console.log(h === j)

        // strict inequality

        console.log(h !== j)

// conditional statements

        // if...else if...else

        // conditional statements are written in block form. using {}

        // if (condtion){

        //     statement to execute

        // }

        // else{

        //     statement to execute

        // }

        let g = 5

        if (x == 5) {

            console.log("number is 5")

        } else {

            console.log("number is not 5")

        }

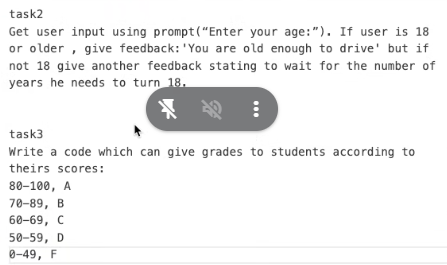
// formatted string

        // a dollar sign $ and backticks is used.

        console.log(`number is ${g}`)

task1 on slide 66

task3



Task4



**Looping**

Basic structure of a loop

1. Initialization – starting point
2. Condition -
3. Update- updates values in the loop

**For loop**

   for (let index = 0; index < array.length; index++) {

            const element = array[index];

        }

**Functions**

Syntax – function function\_name() {

Block of code

}

Function call

Function\_name()

  // ARROW FUNCTIONS

        // -shorter way to write functions in Javascript

        // bellow is a normal function

        const add\_numbers = (nu1,nu2) =>{

            return nu1 + nu2

        }

        // if you have a function that only does one thing, this is how one can do.

        // return here is implicit- implied

        const add\_numbers2 = (num1,num2) => num1 + num2

        sum2 = add\_numbers2(1,2)

        // call back functions

        function greet (name,callback){

            console.log(`Hello ${name}`)

            callback()

        }

        function sayGoodbye(){

            console.log('Goodbye')

        }

        // the fuction is called inside the greet function

        greet('alice',sayGoodbye)

        // i.e do research on call back function. used mostly in javascript frameworks

        // research on: asynchronous and synchronous

        // FUNCTION Expression

        // is when you define a function inside an expression ussually when asigning it to a variable

        // a simple fuction looks like this

        // reaserch on : hoisting

        // function great(){

        //     console.log('hello')

        // }

        // function expression

        const great = function(){

            console.log('hello')

        }

        great()

**FORM INPUT**

// JS helps us access and manipulate form inputs real time

        // js can read user input,validate user input, dynamically update pages based on input

        // Process

        // 1. user fills Form

        // 2. js reads input values

        // 3. js processes the data (calculations,modifications)

        // document.getElementById() - method used to select and access a single html element by its unique id attribute

1. .innerHTML = full HTML content inside a tag/Element

2. .innerTEXT = visible text inside an element

3. .value = only accesses user input in forms only