Javascript

- orginal Name : Livescript
- created by - brendan Eich for Netscape
- renamed : Javascript - "as java was famous then ,to gain popularity"
- microsoft developed their own version of js, known as typescript
- both netscape and microsoft handed over dialup to
- ECMA : Organization which standardizes the language
- now language known as ECMA script (ES) , current ES-7, ES-8 to come
- dynamically typed language

Typescript

- 1. language which is developed on top of javascript is typescript
- 2. initiative taken by Microsoft to implement latest versions (ES7+) of ECMAScript
- 3. better version javascript / type js , added feature on js
- 4. new features
    - data types
    - compiled
    - introduction to class
    - statically typed language

- superClass of JavaScript

- 5. to install typescript on your pc

```
npm install -g typescript
```

- 6. to test if ts is installed

```
tsc -v or tsc --version
```

- 7. for reference , C prog compilation we use (gcc compiler)
    - e.g page1.c---> .o/.out/.exe (executable)(ASM)

Hello world (for ts compilation)

- we use

```
 tsc page1.ts
```

- it creates an executable as a js file (page1.js) so this process is known **transpile** (translate _+ compile)

- now we can run ts file as

```
node page1.js
```

language Fundamentals

1. **Variables**

- 2 ways

1. **implicit variable declaration**

- similar way to declare a variabe in JS
- all datatypes are inferred
- datatype will be implicitely/dynamically assigned
- e.g

```
// number
 const num = 100
 const salary = 10.56
 //string
 const name = 'suraj'
 cosnt address = `address`
 const name = "suraj"
 //object
 //boolean
```

2. **explicit declaration**

- declaring a variable wit required data type
- syntax
- <const/let> <var name>: <data type> = <intial value>
- e.g

```
 const num: number = 100
```

datatypes supported

- **Data Types**
  - all data type in JS will be inferred
  - the datatype will be decided by JS by inspecting the current value in the variable

- **types**

  1. **number**
  - represents whole number and floating point/decimal numbers
  - e.g.

```
//number
 let num:number = 100
 console.log('num ='+ num)
 console.log('data type of num = ' + typeof(num))

 // number
 let salary:number = 10.50
 console.log('datatype of salary =' + typeof(salary));
```

  2. **string**
  - collection of characters
  - string can be declared using
  - single ('..')
  - double ("..")
  - backspace (`) for multiple lines
  - e.g.

```
 // string
 let firstName:string = 'steve'
 console.log('firstName = ' + firstName)
 console.log('datatype of firstName ' + typeof(firstName))

 //

  let address:string = `address line 1,
                  address line 2,
                  address line 3 `
  console.log('address = ' + address)
  console.log('datatype of address =' + typeof(address))
```

  3. **boolean**
  - represent only true or false values
  - e.g

```
 //boolean
   let canVote:boolean = true
   console.log('canVote = '  + canVote + typeof(canVote))
```

4. **undefined**
   - in JS undefined is both : datatype as well as pre-defined value
   - represents a variable without an initial value
   - e.g.

```

   //undefined

   let myvar:undefined
   console.log('myvar = ' + myvar) //undefined
   console.log('datatype of myvar = ' + typeof(myvar))
//undefined

```

5. **object**
   -

```

      const person:{name:string,age:number} = {name: "perosn1",
age : 40}

      console.log(`perosn = ${person}, type of
${typeof(person)}`)
      console.log(person)

```

6. **union**
   - new datatype added by ts
   - datatype with mixture of values,
   - dynamically variable can store any one of the data types
   - e.g

```
 let num: string | number
 num = 100
 num = "test"
 num = true //not ok
```

7. **any**
   - new datatype added by ts
   - any type of value can be stored in the variable

```
    // to store any type of value use any
      let myvar3:any

      myvar3 = 100
      myvar3 = "jea"
    myvar3 = true
```

object oriented programming

- to create a class use **class** keyword

```
class Person{
     //implicit property declaration
    name
    //explicit propery declaration
    age:number
    email:string
    address:string

}

//const p1 = new Person()
//p1.name = "suraj"
//p1.age = 24
//p1.email = "suraj@gmail.com"
//p1.address = "nasik"
//console.log(p1)
```

- **class**
- template to create a an object
- **object**
- collection of key-value pairs, few values are function , few values are properties
- i.e collection of
    - 1. property
    - attributed pf a class e.g. name attribute of a person
    - 2. method
    - function declared inside a class
    - e.g
    - types
    - **setter**
        - used to set value of a property

```
    class Mobile{

      private model:string
      //setter
      public setModel(model:string)
      {
          this.model = model
      }
    }
```

- **getter**
- used to get a value of a property by function return

```
class Mobile{

 private model:string

 //getter
  public getModel()
  {
      return this.model
  }
}
```

- **constructor**
- use to initialize an object, while construction
- in typescript ,the constructor of a class must have a name
- 2 types
    - parameterized constructor
    - parameterless constructor as **constuctor**

```
//parameterless
    class Person{

      constructor ()
      {
        console.log('inside constructor)
      }
    }

    //trying to implement a constructor , use keyword constructor
     public constructor(name:string = '',age:number = 0)
     {
         console.log("inside constructor")
         this._name = name
         this._age = age
     }
```

-destructor // not supported

  - **facilitator***
  - its a utility function , to add facility e.g.

```
    // facilitator or utility
     public canVote()
     {
         if(this._age >= 18)
         { console.log(`${this._name} is eligiable`)}
         else
          {console.log(`${this._name} is not eligiable`) }
     }
```

- access specifiers

1. public

- by default every class as treated as public

2. private
3. protected

- **this** reference
  - using this while accessing the data member are mandetory