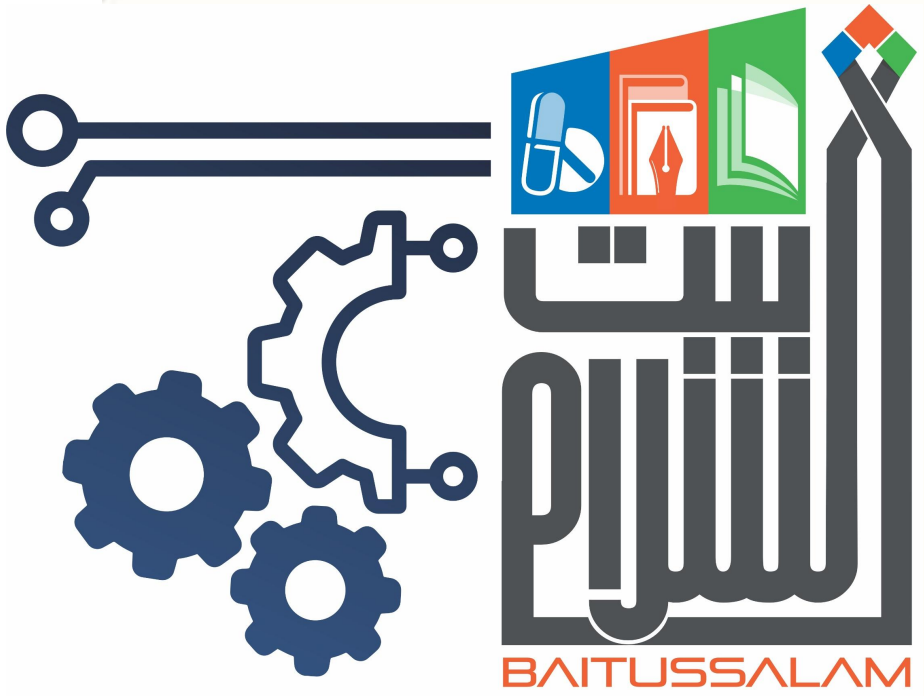

BAITUSSALAM
—TECH PARK—

Class Agenda

BAITUSSALAM
—TECH PARK—

Set, Map, Browser's APIs, and Canvas

Introduction to Browser APIs

Examples of device features and sensors that can be accessed

1. Geolocation
2. Camera and Microphone
3. Device Status
4. Battery Status
5. Clipboard API

Window Navigator

The **navigator object** contains information about the browser.

```
Navigator {vendorSub: '', productSub: '20030107', vendor: 'Google Inc.', maxTouchPoints: 0, scheduling: Schedul
ing, …} i
  appCodeName: "Mozilla"
  appName: "Netscape"
  appVersion: "5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari
  bluetooth: Bluetooth {}
  clipboard: Clipboard {}
  connection: NetworkInformation {onchange: null, effectiveType: '4g', rtt: 150, downlink: 6.35, saveData: fals
  cookieEnabled: true
  credentials: CredentialsContainer {}
  deprecatedRunAdAuctionEnforcesKAnonymity: false
  deviceMemory: 8
  doNotTrack: null
  geolocation: Geolocation {}
  gpu: GPU {wgslLanguageFeatures: WGSLLanguageFeatures}
  hardwareConcurrency: 8
  hid: HID {onconnect: null, ondisconnect: null}
  ink: Ink {}
  keyboard: Keyboard {}
  language: "en-US"
  languages: (2) ['en-US', 'en']
  locks: LockManager {}
  login: NavigatorLogin {}
  managed: NavigatorManagedData {onmanagedconfigurationchange: null}
  maxTouchPoints: 0
  mediaCapabilities: MediaCapabilities {}
  mediaDevices: MediaDevices {ondevicechange: null}
  mediaSession: MediaSession {metadata: null, playbackState: 'none'}
  mimeTypes: MimeTypeArray {0: MimeType, 1: MimeType, application/pdf: MimeType, text/pdf: MimeType, length: 2}
  onLine: true
  pdfViewerEnabled: true
  permissions: Permissions {}
```

Online/Offline Status

Check if the browser is online or offline

```
if (navigator.onLine) {
  console.log("The browser is online.")
} else {
  console.log("The browser is offline.")
}
```

Geolocation API

Allows access to the geographical location of the device

Key method:

- navigator.geolocation.getCurrentPosition()

```
if (navigator.geolocation) {
  navigator.geolocation.getCurrentPosition((position) => {
    console.log("latitude", position.coords.latitude)
    console.log("longitude", position.coords.longitude)
  })
} else {
  console.log("Geolocation is not supported by this browser.")
}
```

Camera and Microphone Access

Access the device's camera and microphone

Key method:

navigator.mediaDevices.getUserMedia()

```
navigator.mediaDevices
  .getUserMedia({ video: true, audio: true })
  .then((stream) => {
    let video = document.querySelector("video")
    video.srcObject = stream
    video.play()
  })
  .catch((err) => {
    console.error("Error accessing media devices.", err)
  })
```

Battery Status API

Provides information about the device's battery status

Key method:

- navigator.getBattery()

```
navigator.getBattery().then((battery) => {
  console.log("Battery level: " + battery.level * 100 + "%")
  console.log("Charging: " + (battery.charging ? "Yes" : "No"))
})
```

Clipboard API

Interact with the clipboard to copy or paste text

Key methods:

navigator.clipboard.writeText()

navigator.clipboard.readText()

```
// Copy text to clipboard
navigator.clipboard.writeText("Welcome to Tech Park!").then(() => {
  console.log("Text copied to clipboard")
})

// Read text from clipboard
navigator.clipboard.readText().then((text) => {
  console.log("Clipboard content: " + text)
})
```

Canvas API

The Canvas API provides a means for drawing graphics via JavaScript and the HTML <canvas> element. Among other things, it can be used for animation, game graphics, data visualization, photo manipulation, and real-time video processing.

Drawing Shapes

Drawing rectangles, circles, and lines

```
const canvas = document.getElementById("myCanvas")
const ctx = canvas.getContext("2d")

// Draw a rectangle
ctx.fillStyle = "blue"
ctx.fillRect(50, 50, 150, 100)

// Draw a circle
ctx.beginPath()
ctx.arc(300, 100, 50, 0, Math.PI * 2, false)
ctx.fillStyle = "red"
ctx.fill()

// Draw a line
ctx.beginPath()
ctx.moveTo(400, 50)
ctx.lineTo(500, 150)
ctx.stroke()
```

Drawing Text

Drawing Text on Canvas

```
ctx.font = "30px Arial"
ctx.fillStyle = "purple"
ctx.fillText("Hello Canvas", 100, 200)
```

Animation in Canvas

Animation on Canvas

```
let x = 0

function animate() {
  ctx.clearRect(0, 0, canvas.width, canvas.height)
  ctx.fillStyle = "orange"
  ctx.fillRect(x, 100, 50, 50)
  x += 2
  requestAnimationFrame(animate)
}

animate()
```

Sets

Sets in JavaScript are collections of **unique values**, meaning **no duplicates** are allowed.

They provide efficient ways to store and manage distinct elements.

```
const people = new Set()

people.add("John")
people.add("Sara")
people.add("Smith")

console.log("people", people)
```

Set Methods

```
const people = new Set(["John", "Sara"])

// adding element
people.add("Smith")

// removing element
people.delete("John")

// to check the property if exist
console.log(people.has("Sara")) // true
console.log(people.has("Ali")) // false

// size of set
console.log("size", people.size)

console.log("people", people)
```

Iterate over Set

We can iterate over set using forEach and for of loop.

We can also use .values method that give us generators

```
people.forEach((person) => {
  console.log(person)
})

for (let person of people) {
  console.log(person)
}
```

Weak Sets

- It does not allow primitive data types.
- A WeakSet is a collection of objects only.
- Unlike Set, WeakSet holds "weak" references to its objects, meaning they do not prevent garbage collection.
- We can not enumerate WeakSet
- Don’t have .clear method

```
let product = { title: "laptop" }

const weakSet = new WeakSet()

weakSet.add(product)

console.log("weakSet", weakSet)
```

Maps

Map is very similar to set but it has key and values.

In Map key could be any type, array, object etc

The Map object holds key-value pairs and remembers the original insertion order of the keys.

```
const products = new Map([["Printer", 6]])

products.set("Mouse", 10)
products.set("Laptop", 7)
products.set("Headphones", 4)

products.has("Laptop") // true
products.has("Headphones") // 4

console.log("products", products)
```

Iterate over Map

```
products.forEach((value, key) => {
  console.log(`${key}: ${value}`)
})

for (let [key, value] of products) {
  console.log(`${key}: ${value}`)
}
```

Weak Map

- **Non-iterable:** WeakMap is not iterable and cannot be looped over.
- **Objects as Keys:** Only objects can be used as keys.
- **Weak References:** Keys are held weakly, meaning they do not prevent garbage collection.
- **No Size Property:** WeakMap does not have a size property or a clear method.

```
let obj1 = { name: "Alice" }
let obj2 = { name: "John" }

const strongMap = new Map()
const weakMap = new WeakMap()

strongMap.set(obj1, "Engineer")
weakMap.set(obj2, "Designer")

console.log(strongMap.get(obj1)) // 'Engineer'
console.log(weakMap.get(obj2)) // 'Designer'

obj1 = null
obj2 = null
```

The End

BAITUSSALAM
—TECH PARK—