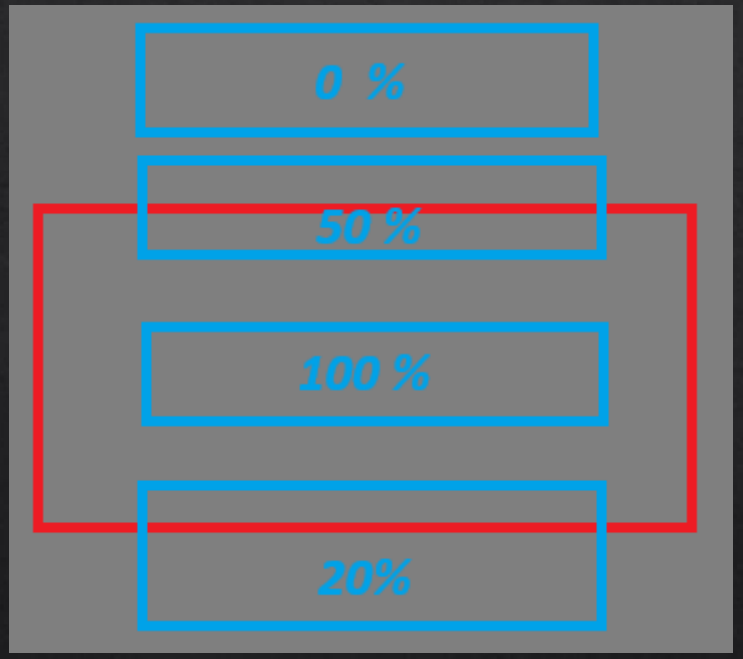
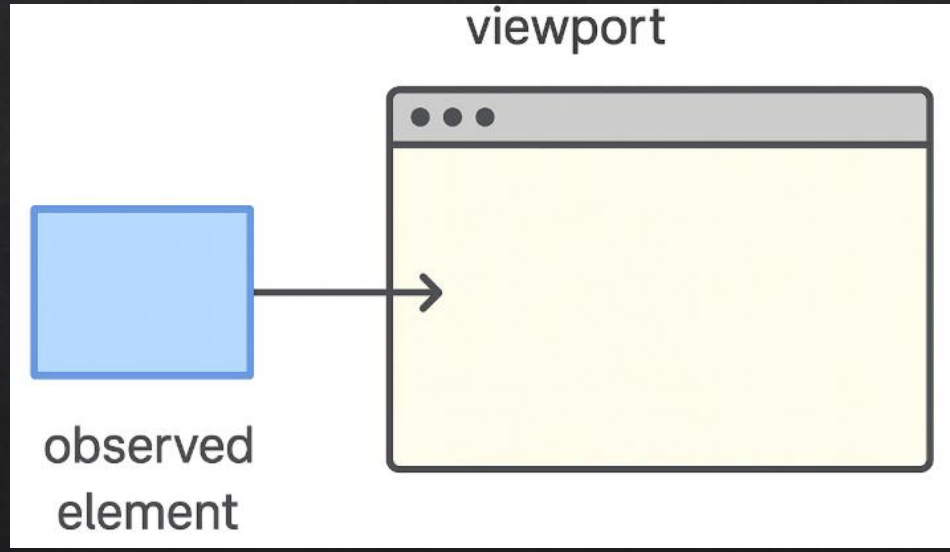
IntersectionObserver





httpsdeveloper.mozilla.orgrudocsWebAPIIntersection_Observer_API

IntersectionObserver - API для асинхронного відстеження перетину (visibility) елементів відносно контейнера (root) або вікна перегляду (viewport) документа.

Застосовується для:

- lazy-loading **(Ліниве завантаження):** Завантаження зображень, відео чи компонентів лише тоді, коли вони потрапляють у зону видимості.
- **Infinite Scrolling (Нескінченний скролінг):** Запит нової порції даних, коли користувач прокручує сторінку до кінця.
- **Відстеження видимості:** Виконання анімації або логіки, коли елемент стає видимим (появі в viewport)
- збору метрик видимості (для аналітики)
- тощо.

Призначення

- Визначити, коли елемент потрапляє/виходить з viewport або контейнера.

- Замінити дорогі scroll/resize обробники з постійними перевірками позицій (краще для продуктивності).

- Забезпечити реактивну поведінку при видимості (lazy-load, анімації, збори аналітики).

Створення та налаштування

IntersectionObserver створюється за допомогою конструктора, який приймає функцію зворотного виклику (callback) і необов'язковий об'єкт налаштувань (options). Функція callback викликається, коли елемент target, що відслідковується, зявляється у області видимості root.

```
const observer = new IntersectionObserver(callback, options);
```

Функція зворотного виклику (`callback`)

`entries` - масив об'єктів відестення,
observer - посилання на сам об’єкт відстежувач

```
function callback(entries, observer) {
    entries.forEach(entry => {
        // 'entry' містить інформацію про перетин
        if (entry.isIntersecting) {
            // Елемент у зоні видимості (перетнув поріг)
            entry.target.classList.add('visible');
            // Можна припинити спостереження, якщо більше не потрібно
            // observer.unobserve(entry.target);
        } else {
            // Елемент поза зоною видимості
            entry.target.classList.remove('visible');
        }
    });
}
```

Налаштування (`options`)

Властивість	Тип	Опис
`root`	`Element` або `null`	Елемент, який використовується як область перегляду (viewport). `null` (за замовчуванням) означає вікно браузера (viewport).
`rootMargin`	`String`	Відступи навколо `root`. Задається як CSS-властивість `margin` (наприклад, `'10px 20px 30px 40px'` або `'50px 0px'`). Дозволяє спрацьовувати спостерігачу **до** або **після** фактичного входу в root.
`threshold`	`Number` або `Array<Number>`	Одне число або масив чисел між `0.0` і `1.0`. **Поріг перетину.** `0.0` означає спрацювання, як тільки з'являється 1 піксель елемента. `1.0` означає спрацювання, коли елемент повністю видимий.

```
const target = document.querySelector('.my-target-element');

// Починаємо спостереження
observer.observe(target);
```

Створення та налаштування

IntersectionObserver створюється за допомогою конструктора, який приймає функцію зворотного виклику (callback) і необов'язковий об'єкт налаштувань (options).

```
const observer = new IntersectionObserver(callback, options);
observer.observe(targetElement);
// ...
observer.unobserve(targetElement);
observer.disconnect(); // зупинити всі спостереження
```

Метод	Опис
`observe(targetElement)`	Починає спостереження за вказаним DOM-елементом.
`unobserve(targetElement)`	Припиняє спостереження за вказаним DOM-елементом.
`disconnect()`	Припиняє спостереження за **всіма** цільовими елементами, пов'язаними з цим observer.
`takeRecords()`	Повертає список об'єктів `IntersectionObserverEntry` для всіх незавершених переходів.

Функція зворотного виклику (`callback`)

```
function callback(entries, observer) {
    entries.forEach(entry => {
        // 'entry' містить інформацію про перетин
        if (entry.isIntersecting) {
            // Елемент у зоні видимості (перетнув поріг)
            entry.target.classList.add('visible');
            // Можна припинити спостереження, якщо більше не потрібно
            // observer.unobserve(entry.target);
        } else {
            // Елемент поза зоною видимості
            entry.target.classList.remove('visible');
        }
    });
}
```

Налаштування (`options`)

Властивість	Тип	Опис
`root`	`Element` або `null`	Елемент, який використовується як область перегляду (viewport). `null` (за замовчуванням) означає вікно браузера (viewport).
`rootMargin`	`String`	Відступи навколо `root`. Задається як CSS-властивість `margin` (наприклад, `'10px 20px 30px 40px'` або `'50px 0px'`). Дозволяє спрацьовувати спостерігачу **до** або **після** фактичного входу в root.
`threshold`	`Number` або `Array<Number>`	Одне число або масив чисел між `0.0` і `1.0`. **Поріг перетину.** `0.0` означає спрацювання, як тільки з'являється 1 піксель елемента. `1.0` означає спрацювання, коли елемент повністю видимий.

Створення та налаштування

IntersectionObserver створюється за допомогою конструктора, який приймає функцію зворотного виклику (callback) і необов'язковий об'єкт налаштувань (options).

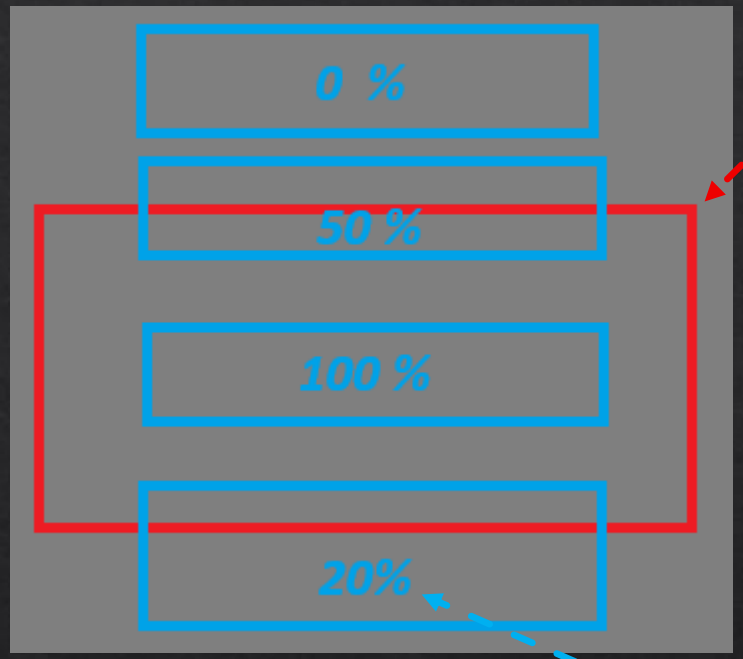
```
const observer = new IntersectionObserver(callback, options);
```

Функція зворотного виклику (`callback`)

```
function callback(entries, observer) {
    entries.forEach(entry => {
        // 'entry' містить інформацію про перетин
        if (entry.isIntersecting) {
            // Елемент у зоні видимості (перетнув поріг)
            entry.target.classList.add('visible');
            // Можна припинити спостереження, якщо більше не потрібно
            // observer.unobserve(entry.target);
        } else {
            // Елемент поза зоною видимості
            entry.target.classList.remove('visible');
        }
    });
}
```

Об'єкт `IntersectionObserverEntry`

Властивість	Опис
`target`	DOM-елемент, за яким ведеться спостереження.
`isIntersecting`	`true`, якщо елемент перетинає `root` (видимий), інакше — `false`.
`intersectionRatio`	Частка перетину (від `0.0` до `1.0`). Наприклад, `0.5` означає, що видно половину елемента.
`rootBounds`	Розміри та позиція кореневого елемента (`root`).
`boundingClientRect`	Розміри та позиція цільового елемента (`target`).



Налаштування (`options`)

Властивість	Тип	Опис
`root`	`Element` або `null`	Елемент, який використовується як область перегляду (viewport). `null` (за замовчуванням) означає вікно браузера (viewport).
`rootMargin`	`String`	Відступи навколо `root`. Задається як CSS-властивість `margin` (наприклад, `'10px 20px 30px 40px'` або `'50px 0px'`). Дозволяє спрацьовувати спостерігачу **до** або **після** фактичного входу в root.
`threshold`	`Number` або `Array<Number>`	Одне число або масив чисел між `0.0` і `1.0`. **Поріг перетину.** `0.0` означає спрацювання, як тільки з'являється 1 піксель елемента. `1.0` означає спрацювання, коли елемент повністю видимий.

Об'єкт `IntersectionObserverEntry`

Властивість	Опис
`target`	DOM-елемент, за яким ведеться спостереження.
`isIntersecting`	`true`, якщо елемент перетинає `root` (видимий), інакше — `false`.
`intersectionRatio`	Частка перетину (від `0.0` до `1.0`). Наприклад, `0.5` означає, що видно половину елемента.
`rootBounds`	Розміри та позиція кореневого елемента (`root`).
`boundingClientRect`	Розміри та позиція цільового елемента (`target`).

Lazy loading зображень

```
const callback = (entries, obs) => {
  entries.forEach((entry) => {
    if (entry.isIntersecting) {
      // Коли елемент потрапив у зону видимості root:

      // 1. Встановлюємо реальне джерело зображення
      entry.target.src = entry.target.dataset.src

      // 2. Прибираємо клас 'lazy' (для зміни стилів)
      entry.target.classList.remove('lazy')

      // 3. Припиняємо спостереження, оскільки зображення завантажено
      obs.unobserve(entry.target)
    }
  })
}
```

```
<div id="scroll-container">

  <img
    class="lazy"
    alt="Image 1"
    data-src="https://picsum.photos/id/10/800/300"
    src=""
  />

</div>
```

```
// Отримуємо кореневий елемент
const scrollContainer = document.getElementById('scroll-container')
```

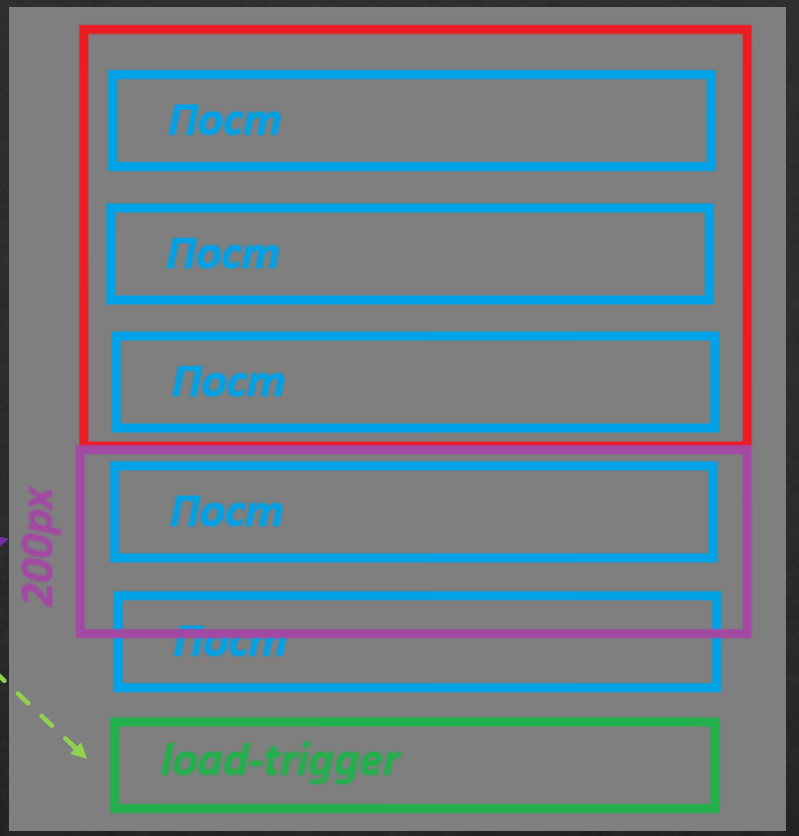
```
const options = {
  // !!! Вказуємо контейнер як кореневий елемент !!!
  root: scrollContainer,
  // Розпочати завантаження за 200px до появи в зоні видимості контейнера
  rootMargin: '0px 0px 200px 0px',
  // 10% елемента має бути видимим, щоб спрацював callback
  threshold: 0.1,
}
```

```
const observer = new IntersectionObserver(callback, options)

//Отримуємо DOM-елементи
const images = document.querySelectorAll('img.lazy')
// Починаємо спостереження за кожним зображенням
images.forEach((img) => observer.observe(img))
```

Infinite scroll (підвантаження контенту)
Коли користувач доходить до кінця списку - підвантажуємо нові дані.

```html
<div>
  <ul id="posts">
    ... тут будуть пости ...
  </ul>
  <div id="load-trigger"></div>
</div>
```




```js
const observer = new IntersectionObserver(
  async (entries) => {
    // Якщо елемент-тригер з'явився у полі зору
    if (entries[0].isIntersecting) {
      // Викликаємо функцію завантаження
      await loadMore()
    }
  },
  {
    // root: null (вікно браузера)
    // rootMargin: '0px 0px 200px 0px' - спрацює за 200px до того, як тригер потрапить у вікно
    rootMargin: '0px 0px 200px 0px',
    threshold: 0.01, // Спрацює, як тільки хоча б 1% елемента тригера видно
  }
)


const trigger = document.querySelector('#load-trigger')

// 2. Починаємо спостереження за тригером
// Це спрацює, коли завантажаться перші пости і тригер підніметься у поле зору
observer.observe(trigger)
```

Анімації при появі елемента у viewport

```css
.fade {
  opacity: 0;
  transform: translateY(30px);
  transition: all 0.6s ease;
}

.fade.visible {
  opacity: 1;
  transform: translateY(0);
}
```

```html
<div class="section">
  <h2 class="slide-in-left">Fade In</h2>
  <div class="demo-box fade">Плавне з'явлення знизу вгору</div>
</div>
```

```js
// Створюємо спостерігач для анімацій
const observer = new IntersectionObserver(
  (entries, obs) => {
    entries.forEach((entry) => {
      if (entry.isIntersecting) {
        // Додаємо клас 'visible' для запуску анімації
        entry.target.classList.add('visible')
        // Припиняємо спостереження після анімації
        obs.unobserve(entry.target)
      }
    })
  },
  {
    threshold: 0.2, // Елемент має бути видимим на 20%
    rootMargin: '0px 0px -50px 0px', // Почати анімацію за 50px до появи
  }
)

// Починаємо спостереження за всіма елементами з анімаціями
elements.forEach((el) => observer.observe(el))
```

Вимірювання видимості реклами

Логує в консоль, коли рекламний блок видно більше ніж на 50%.

```html
<h1>Скроль, щоб побачити рекламу 👇</h1>
<div id="ad">Реклама</div>
```

```js
const adBlock = document.querySelector('#ad')

const observer = new IntersectionObserver(
  (entries) => {
    entries.forEach((entry) => {
      if (entry.intersectionRatio > 0.5) {
        console.log('✅ Реклама видима більше ніж на 50%')
      } else {
        console.log('🚫 Реклама менш ніж на 50% у полі зору')
      }
    })
  },
  { threshold: [0, 0.5, 1] }
)

observer.observe(adBlock)
```

Активний розділ меню

Підсвічує пункт меню, коли відповідна секція видима.

```html
<nav>
  <a href="#home">Головна</a>
  <a href="#about">Про нас</a>
  <a href="#services">Послуги</a>
  <a href="#contact">Контакти</a>
</nav>

<section id="home">Головна</section>
<section id="about">Про нас</section>
<section id="services">Послуги</section>
<section id="contact">Контакти</section>
```

```js
const sections = document.querySelectorAll('section')
const navLinks = document.querySelectorAll('nav a')

const observer = new IntersectionObserver(
  (entries) => {
    entries.forEach((entry) => {
      if (entry.isIntersecting) {
        navLinks.forEach((link) => link.classList.remove('active'))
        const id = entry.target.getAttribute('id')
        document
          .querySelector(`nav a[href="#${id}"]`)
          .classList.add('active')
      }
    })
  },
  { threshold: 0.6 }
)

sections.forEach((sec) => observer.observe(sec))
```

Автоматична пауза відео

Відео відтворюється лише тоді, коли його видно щонайменше на 50%.

```html
<h1 style="text-align: center">Скроль, щоб керувати відео</h1>

<video muted loop>
  <source
    src="https://interactive-examples.mdn.mozilla.net/media/cc0-videos/flower.mp4"
    type="video/mp4"
  />
</video>
```

```js
const videos = document.querySelectorAll('video')

const observer = new IntersectionObserver(
  (entries) => {
    entries.forEach((entry) => {
      if (entry.isIntersecting) {
        entry.target.play()
        console.log('▶️ Відтворення відео')
      } else {
        entry.target.pause()
        console.log('⏸ Пауза відео')
      }
    })
  },
  { threshold: 0.5 }
)

videos.forEach((v) => observer.observe(v))
```

ResizeObserver

ResizeObserverResizeObserver - це інтерфейс Web API, який надає механізм для асинхронного спостереження за змінами розмірів елементів DOM. На відміну від обробника подій window.onresize, який спрацьовує лише при зміні розмірів вікна браузера, ResizeObserver дозволяє відстежувати зміни розмірів будь-якого елемента внаслідок CSS-анімації, маніпуляцій DOM або дії користувача (наприклад, зміна розміру контейнера resize: both).

```
const observer = new ResizeObserver(callback);
```

Функція зворотного Виклику (callback) - ця функція викликається асинхронно після кожного перемальовування браузером, коли розмір спостережуваного елемента змінився.

entries - масив об'єктів відестення, чий розмір змінився в поточному циклі
observer - посилання на сам об’єкт відстежувач

```
const observerCallback = (entries, observer) => {
  for (let entry of entries) {
    // Логіка обробки зміни розміру
    const { width, height } = entry.contentRect
    console.log(
      `Елемент ${entry.target.id} має нові розміри: ${width}x${height}`
    )
  }
}
```

Властивість	Тип	Опис
`target`	`Element`	DOM-елемент, розмір якого змінився.
`contentRect`	`DOMRectReadOnly`	Об'єкт, що містить нові розміри та позицію цільового елемента. Основними властивостями є `width` і `height`.
`contentBoxSize`	`Array<ResizeObserverSize>`	Розміри області контенту, виміряні в пікселях. Виключає відступи (`padding`), рамки (`border`) та поля (`margin`).
`borderBoxSize`	`Array<ResizeObserverSize>`	Розміри області рамки, виміряні в пікселях. Включає відступи (`padding`) та рамки (`border`).
`devicePixelContentBoxSize`	`Array<ResizeObserverSize>`	Розміри області контенту у пікселях пристрою (корисно для високої роздільної здатності).

```
// 2. Створення ResizeObserver
const observer = new ResizeObserver(observerCallback)

// 3. Запуск спостереження за обома цільовими елементами
observer.observe(box) // Спостерігаємо за елементом, який змінює користувач
observer.observe(viewportBox) // Спостерігаємо за елементом, який змінюється з вікном
```

Метод	Опис
`observe(targetElement)`	Починає спостереження за вказаним DOM-елементом (`targetElement`).
`unobserve(targetElement)`	Припиняє спостереження за конкретним елементом.
`disconnect()`	Припиняє спостереження за **всіма** елементами, які були прив'язані до цього спостерігача.