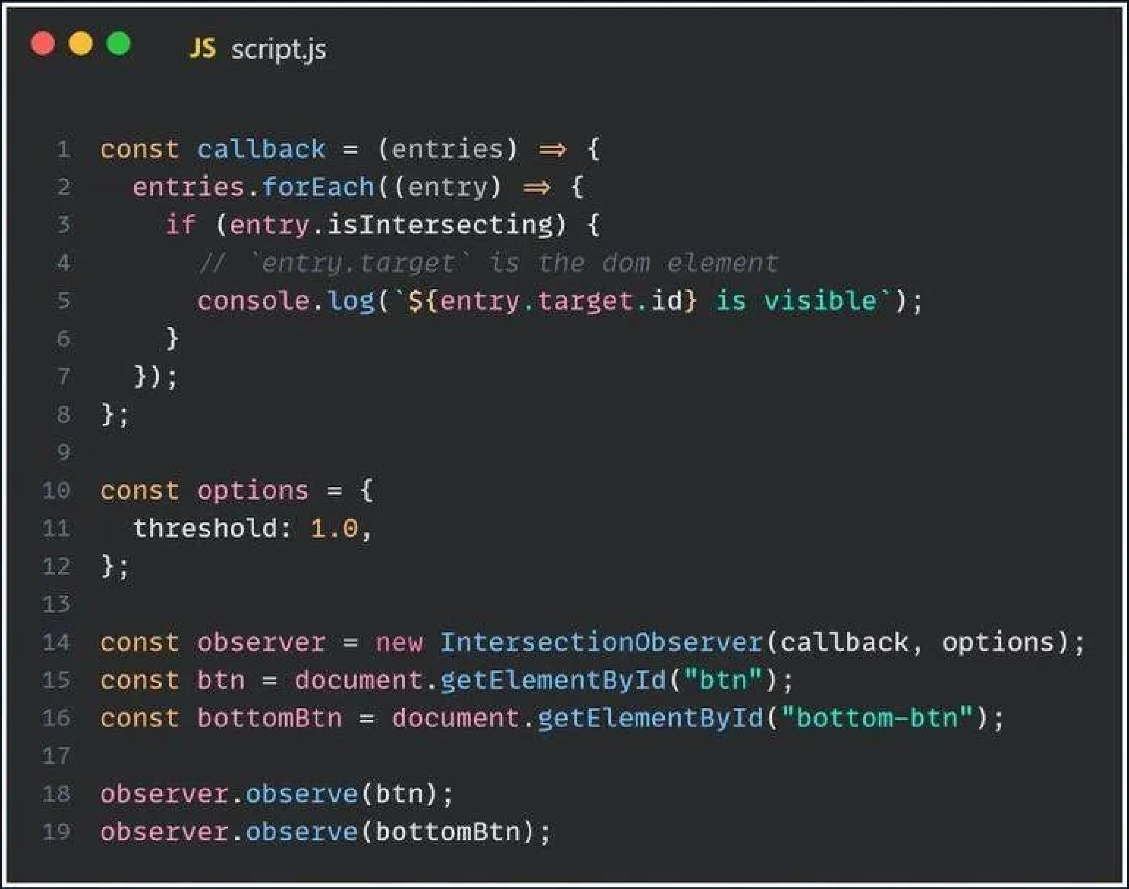
@pyplane_code

6 Killer Functions In JavaScript

Check if an element is visible in the viewport

IntersectionObserver is a great way to check if an element is visible in the viewport.



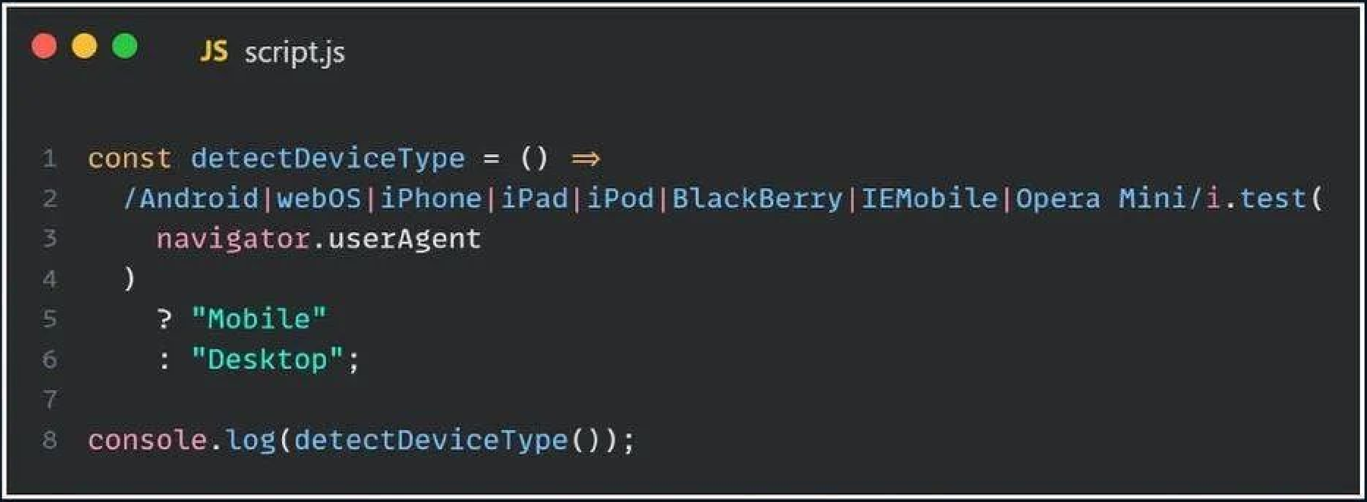
JS script.js

```javascript
const callback = (entries) => {
  entries.forEach((entry) => {
    if (entry.isIntersecting) {
      // `entry.target` is the dom element
      console.log(`${entry.target.id} is visible`);
    }
  });
};

const options = {
  threshold: 1.0,
};

const observer = new IntersectionObserver(callback, options);
const btn = document.getElementById("btn");
const bottomBtn = document.getElementById("bottom-btn");

observer.observe(btn);
observer.observe(bottomBtn);
```

You can customize the behavior of the observer using the ***option*** parameter. ***threshold*** is the most useful attribute, it defines **the percentage of the element that needs to be visible in the viewport for the observer to trigger.**

Detect device

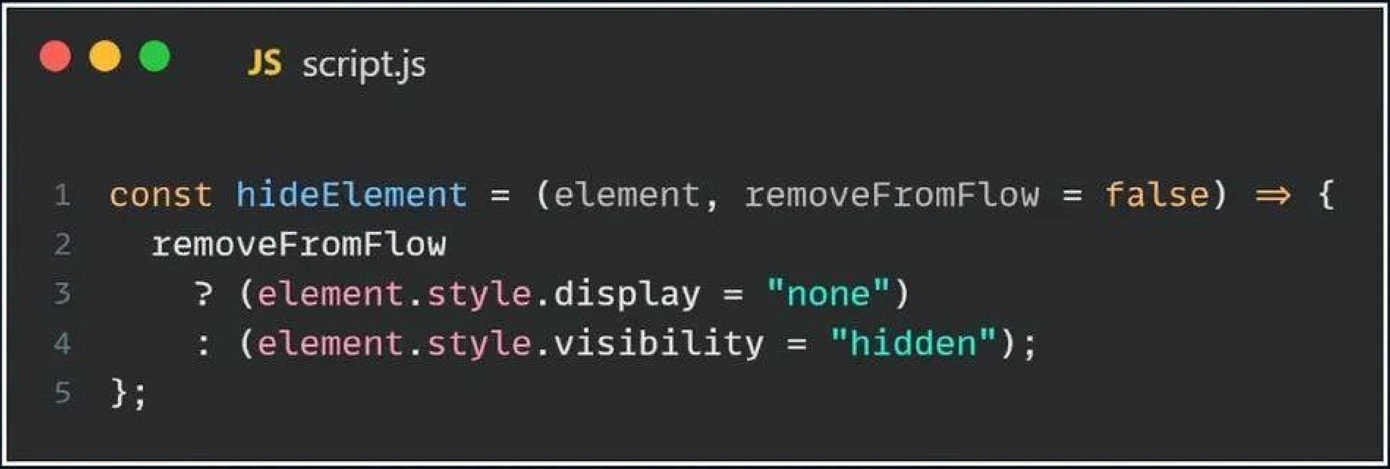
You can use the ***navigator.userAgent*** to gain minute insights and **detect the device running the application**



```javascript
const detectDeviceType = () =>
  /Android|webOS|iPhone|iPad|iPod|BlackBerry|IEMobile|Opera Mini/i.test(
    navigator.userAgent
  )
    ? "Mobile"
    : "Desktop";

console.log(detectDeviceType());
```

Hide elements

You can just **toggle the visibility** of an element using the ***style.visibility*** property and in case you want to **remove it from the render flow**, you can use the ***style.display*** property.



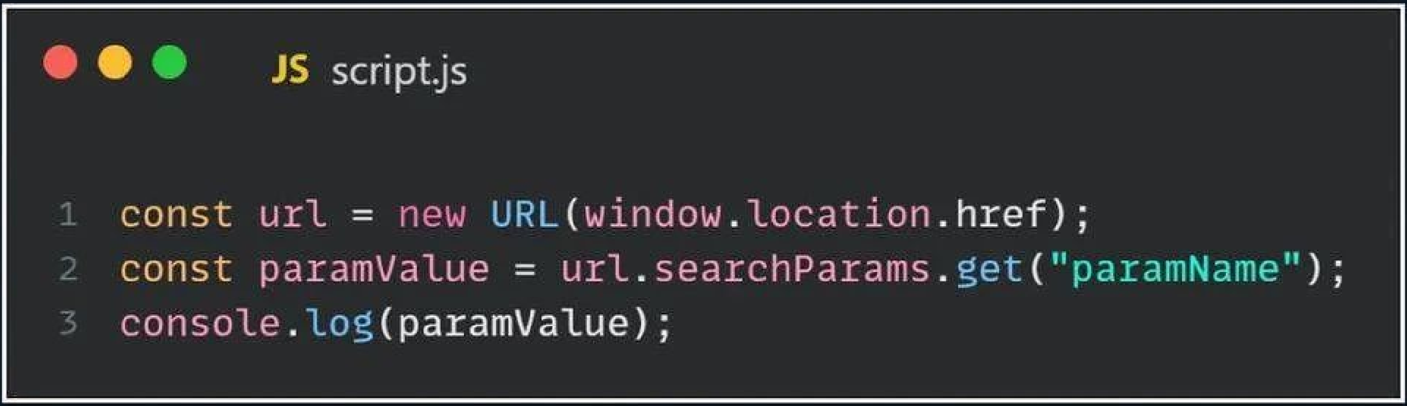
JS script.js

```javascript
const hideElement = (element, removeFromFlow = false) ⇒ {
  removeFromFlow
    ? (element.style.display = "none")
    : (element.style.visibility = "hidden");
};
```

If you don't remove an element from the render flow, it will be hidden, but **its space will still be occupied**. It is highly useful while rendering long lists of elements, the elements NOT in view (can be tested using ***IntersectionObserver***) can be **hidden to provide a performance boost.**

Get the parameters from the URL

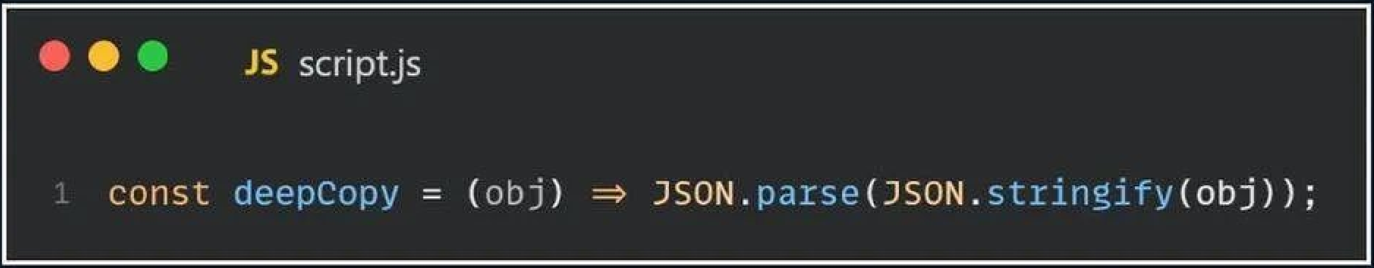
JavaScript makes fetching the ***parameters*** from any address a walk in the park using the ***URL*** object.


JS script.js

```javascript
const url = new URL(window.location.href);
const paramValue = url.searchParams.get("paramName");
console.log(paramValue);
```

Deep copy an object with ease

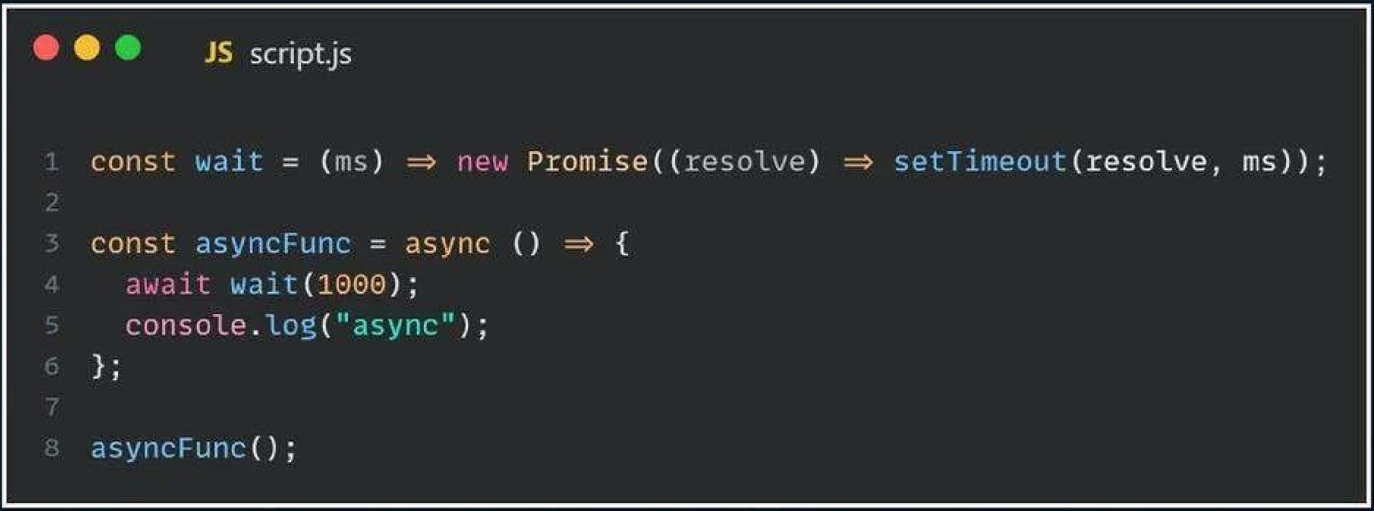
You can ***deep copy*** any object by **converting it to a string and back to an object.**


JS script.js

```js
const deepCopy = (obj) => JSON.parse(JSON.stringify(obj));
```

wait function

JavaScript does ship with a ***setTimeout*** function, but it does not return a ***Promise*** object, making it hard to use in ***async functions***. So we have to write our own ***wait/sleep*** function.


JS script.js

```javascript
const wait = (ms) => new Promise((resolve) => setTimeout(resolve, ms));

const asyncFunc = async () => {
  await wait(1000);
  console.log("async");
};

asyncFunc();
```