🔒

null and undefined

- Both `null` and `undefined` are datatype as well as values that represent nothingness. Hence they are often called non-value values.
- Many languages use only one non-value values, often `null`. But JavaScript uses two.

Undefined

- A variable can be declared primarily in following ways:

```
//global
a=1;

//function scoped
var a = 1;

//function and block scoped
let a =1;
const a =1;
```

- But if we try to access a variable without defining it

```
b
console.log(b)
```

- Then we will get `ReferenceError`, here we are trying to access a variable(b) that has not been defined.
- `undefined` is different from a variable that **has not been defined**.
- In JavaScript undefined is a primitive datatype.
- It is a property on the global object, hence, it is available in global scope.

```
window.hasOwnProperty("undefined") //true
```

- We can't change the value of this property, even if we try to change, our attempt will be ignored

```
window.undefined =  1;
console.log(window.undefined)//undefined
```

- Also since undefined is present in the global scope, we can declare variable with the name **undefined** in a local scope, but it is not recommended, because of readability reasons.

```
function myFunction(){
  let undefined = 1;
  console.log(undefined);
}

myFunction();
//output 1 on the console.
```

- A regular function will return `undefined` if nothing is returned explicitly.

```
function add(a,b){

}

console.log(add(1,2))//undefined
```

- If no argument is passed to a function, `undefined` is passed as default.

```
function add(a,b){
  console.log(b)//undefined
}

add(1);
```

- Also if we pass undefined, the default parameter is taken in the function.

```
function add(a,b=1){
  console.log(b) //1
}

add(1,undefined)
```

- Any other value except `undefined` will not behave as above.
- Always use strict equality(===) and not standard equality(==) with `undefined`, because standard equality treats both `undefined` and `null` as same.

```
var x;

if(x==undefined){
  console.log("this statement will be executed")
}
```

```
var x = null
if(x==undefined){
  console.log("this statement will also be executed")
}
```

```
var x;

if(x===undefined){
  console.log("this satement will be executed")
}
```

```
var x=null;
if(x===undefined){
  cosole.log("this statement won't be executed");
}
```

Checking if a variable has been declared or not

- Using `typeof` operator

```
//x has not been declared yet
if(typeof x === "undefined"){
  //this line runs
}
```

- `typeof` doesn't throw `ReferenceError` even if the variable has not been declared.
- The following way will throw an error, and hence not recommended

```
//x has not been declared yet
if(x === "undefined"){
  //this line won't run
}
```

- Another way to check if a variable has been defined is by using `in` operator. The following method doesn't work for local scope variables.

```
if(x in window){
  //this line runs if x exists in the global scope
}
```

- `hasOwnProperty` can also be used to check if a variable has been declared or not.

Undefined and Execution Context

- Following variable(a) will have `undefined` assigned to it.

```
let a;
console.log(a)//undefined
```

- But why?
- When the global execution context is created, the engine first goes through creation phase.
- In creation phase, all variables, except the ones that are inside a function are given memory in the heap. At this point of time, these variables store `undefined`.

- `undefined` is a data type in JavaScript that indicates that memory has been allocated for a variable, but nothing has been assigned to that variable yet.
- Assignment is done in the **execution phase.** If a variable has not been assigned anything, during the runtime, it will stay in the undefined state, throughout the program**.**
- `typeof undefined` returns `undefined`

Good to do

- It is a good practice to not assign `undefined` to variables yourself. If you want to indicated that a variable doesn't contain anything, consider assigning it `null`. This makes it clear for other people reading your code ,that which variables has been purposefully cleared and which were not.

null

- `null` is a primitive data-type in JS
- It is generally used by the programmer to show that a variable has no value. The programmer has purposefully cleared it. It is different from `undefined`, in which no value has been assigned even a single time.
- `null` and `undefined` both are treated as falsy for boolean operations.
- `typeof null` returns object, but why?
- Douglas Crockford thinks its a mistake.
- Kiro Risk thinks it was intentional:
    - The reasoning behind this is that `null` in contrast with `undefined` was (and still is) often used where objects appear. In other words, `null` is often used to signify an empty reference to an object. When Brendan Eich created JavaScript, he followed the same paradigm, and it made sense (arguably) to return "object". In fact, the ECMAScript specification defines `null` as *the primitive value that represents the intentional absence of any object value*(ECMA-262, 11.4.11).
- Even though `typeof null` is object, it is a primitive data type.

Miscellaneous

- `undefined` == `null` returns `true`, because specs says that, and also it makes kind of sense, since they both represent **nothing**.
- `undefined` === `null` returns `false` , since both don't have same type.
- `isNaN(1+null)` returns `false`
- `isNaN(1+undefined)` returns `true`

References

- https://www.youtube.com/watch?v=B7iF6G3EyIk(video by Akshay Saini)
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined(MDN)
- https://medium.com/weekly-webtips/null-and-undefined-in-javascript-d9bc18acdaff(article by Megan Lo)
- https://medium.com/@stephenthecurt/a-brief-history-of-null-and-undefined-in-javascript-c283caab662e(article by Stephen Curtis)
- https://stackoverflow.com/a/18808300/16716190(an answer by chryss)
- https://levelup.gitconnected.com/the-non-value-trio-of-javascript-db609004e3ec(article by Joseph Pyram)