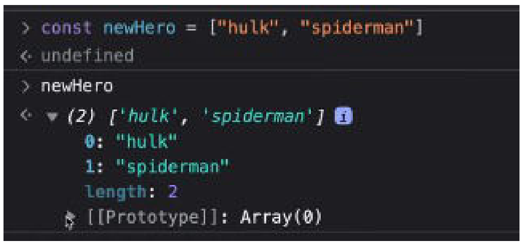
Prototype in JS

new keyword:

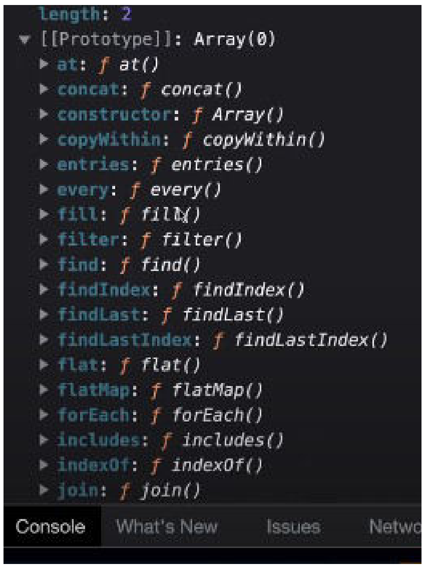
- Default behaviour of JS is prototypal.(access parent, grand parent... till it find the desired thing).

- this, new, classes ,prototypal inheritance comes from prototype.



```
> const newHero = ["hulk", "spiderman"]
< undefined
> newHero
< ▾ (2) ['hulk', 'spiderman'] i
      0: "hulk"
      1: "spiderman"
      length: 2
    ▸ [[Prototype]]: Array(0)
```

- if we expand the prototype (Array) we will get some methods, that comes with array

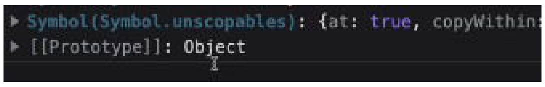


```
    length: 2
  ▾ [[Prototype]]: Array(0)
    ▸ at: ƒ at()
    ▸ concat: ƒ concat()
    ▸ constructor: ƒ Array()
    ▸ copyWithin: ƒ copyWithin()
    ▸ entries: ƒ entries()
    ▸ every: ƒ every()
    ▸ fill: ƒ fill()
    ▸ filter: ƒ filter()
    ▸ find: ƒ find()
    ▸ findIndex: ƒ findIndex()
    ▸ findLast: ƒ findLast()
    ▸ findLastIndex: ƒ findLastIndex()
    ▸ flat: ƒ flat()
    ▸ flatMap: ƒ flatMap()
    ▸ forEach: ƒ forEach()
    ▸ includes: ƒ includes()
    ▸ indexOf: ƒ indexOf()
    ▸ join: ƒ join()
```

- prototypal behaviour means never lose, what else i can do.

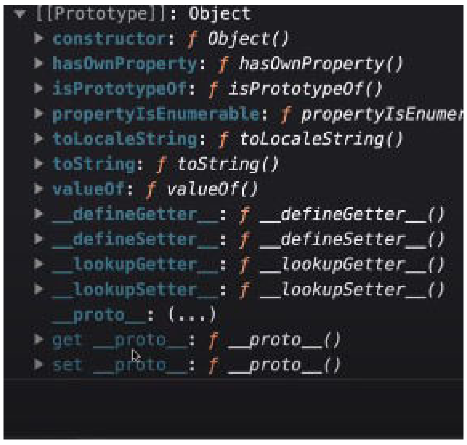
- **prototypal inheritance** : If we didn't find here, we will go to parent if not there then grandparent. ...

- if we scroll till last we will get another prototype(object prototype)



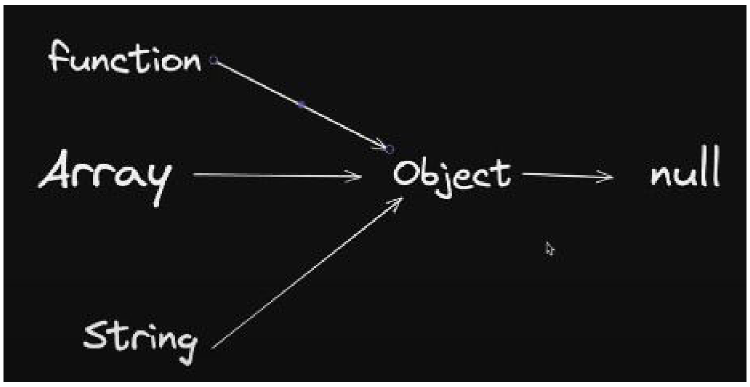
```
▸ Symbol(Symbol.unscopables): {at: true, copyWithin:
▸ [[Prototype]]: Object
```

we we expand this prototype we will get :



```
▼ [[Prototype]]: Object
  ▶ constructor: ƒ Object()
  ▶ hasOwnProperty: ƒ hasOwnProperty()
  ▶ isPrototypeOf: ƒ isPrototypeOf()
  ▶ propertyIsEnumerable: ƒ propertyIsEnumer
  ▶ toLocaleString: ƒ toLocaleString()
  ▶ toString: ƒ toString()
  ▶ valueOf: ƒ valueOf()
  ▶ __defineGetter__: ƒ __defineGetter__()
  ▶ __defineSetter__: ƒ __defineSetter__()
  ▶ __lookupGetter__: ƒ __lookupGetter__()
  ▶ __lookupSetter__: ƒ __lookupSetter__()
    __proto__: (...)
  ▶ get __proto__: ƒ __proto__()
  ▶ set __proto__: ƒ __proto__()
```

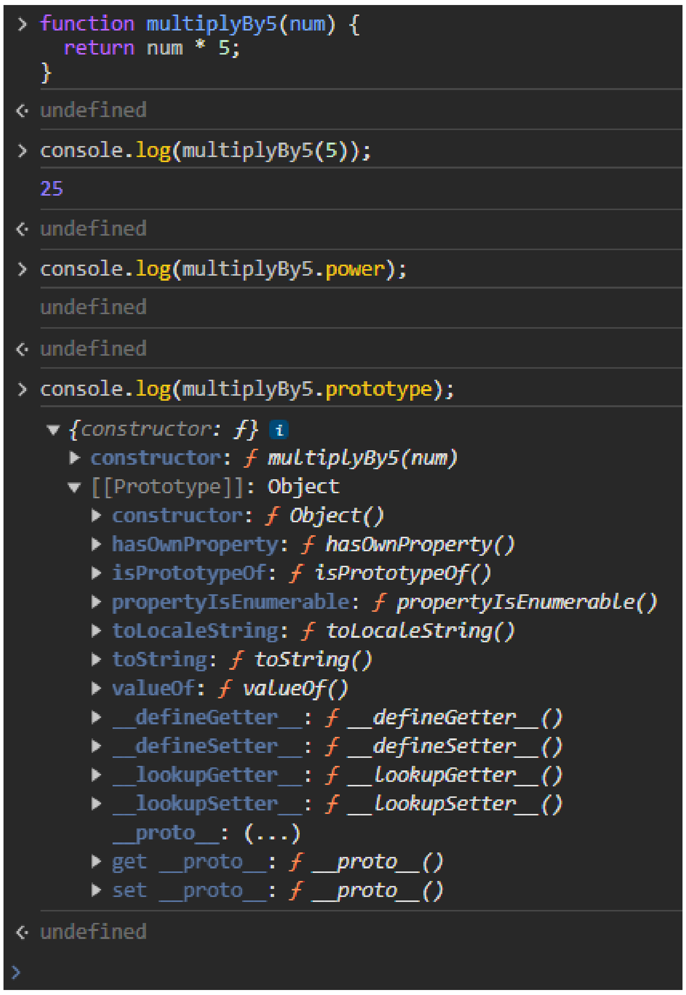
- but now we didn’t see any prototype.


$$\begin{tikzcd}
\text{function} \arrow[dr] & & \\
\text{Array} \arrow[r] & \text{Object} \arrow[r] & \text{null} \\
\text{String} \arrow[ur] & &
\end{tikzcd}$$

Array/string prototype is object, but object protype is none so null.

- In JS everything is an object , whatever property object have array/object inherit the same property.

- even function in JS are object.



```
> function multiplyBy5(num) {
    return num * 5;
  }
< undefined
> console.log(multiplyBy5(5));
  25
< undefined
> console.log(multiplyBy5.power);
  undefined
< undefined
> console.log(multiplyBy5.prototype);
  ▼ {constructor: ƒ} i
    ▶ constructor: ƒ multiplyBy5(num)
    ▼ [[Prototype]]: Object
      ▶ constructor: ƒ Object()
      ▶ hasOwnProperty: ƒ hasOwnProperty()
      ▶ isPrototypeOf: ƒ isPrototypeOf()
      ▶ propertyIsEnumerable: ƒ propertyIsEnumerable()
      ▶ toLocaleString: ƒ toLocaleString()
      ▶ toString: ƒ toString()
      ▶ valueOf: ƒ valueOf()
      ▶ __defineGetter__: ƒ __defineGetter__()
      ▶ __defineSetter__: ƒ __defineSetter__()
      ▶ __lookupGetter__: ƒ __lookupGetter__()
      ▶ __lookupSetter__: ƒ __lookupSetter__()
        __proto__: (...)
      ▶ get __proto__: ƒ __proto__()
      ▶ set __proto__: ƒ __proto__()
< undefined
>
```

We can see the function is also an object and it have prototype as well.

- prototype gives not only the methods, it also give the some internal property

-- The output of `console.log(multiplyBy5.prototype);`

gives us { },so it’s empty.
When we do this.name = name. reference get stored in { } ie,
{ this.name }