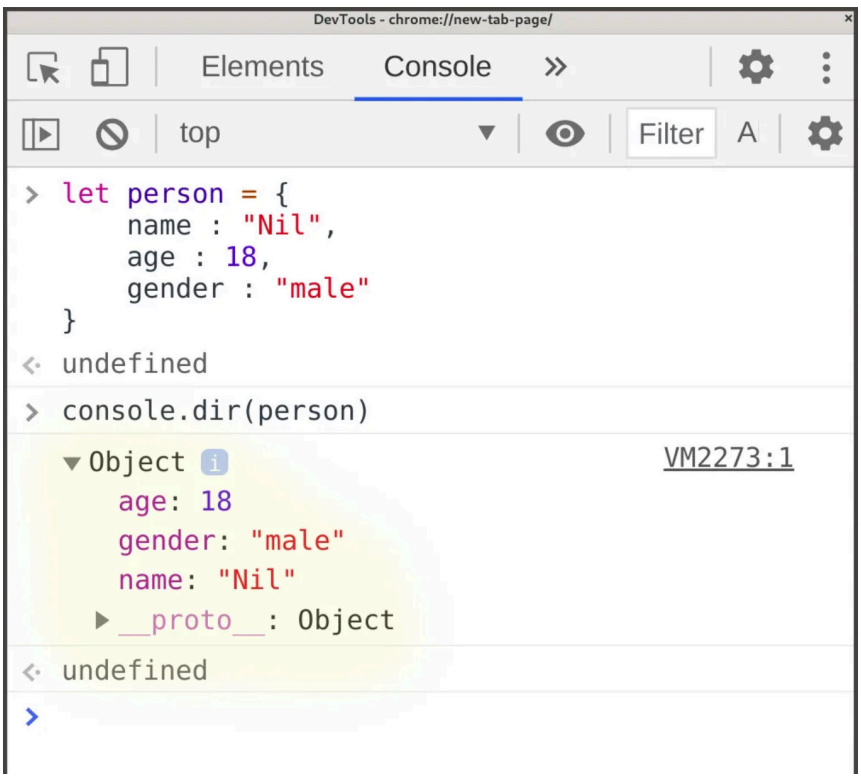
# PROTOTYPE

→ Prototype are the machanism by which javascript object inherit feature from one to another.

→ In javascript everything is object so, you easily able to use prototype with string, array function and so on...



## Explain the difference between `__proto__` and `prototype` ?

`__proto__` is used to **access an object's prototype directly**, while `prototype` is used to define the prototype for objects created with a constructor function.

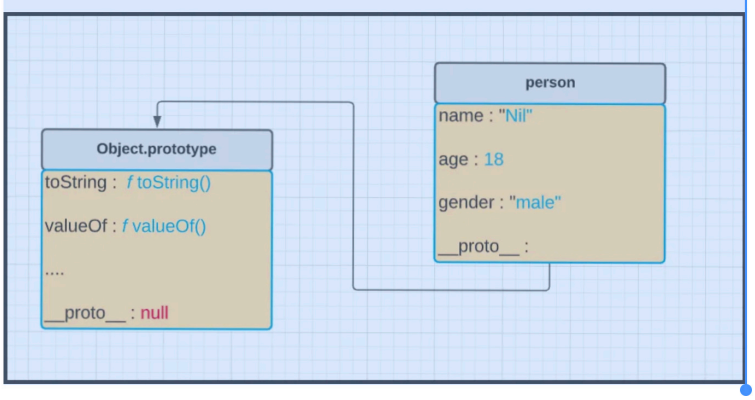
It's important to note that **`prototype` is a property of functions** (specifically, constructor functions), while **`__proto__` is a property of objects** themselves.

Two distinct properties that are related to prototypal inheritance:

The `__proto__` property is a default property added to every object. This property points to the `prototype` of the object.

The default `prototype` of every object is `Object.prototype`. Therefore, the `__proto__` property of the `person` object points to the `Object.prototype`.

If whatever I said is actually true then the illustration would look something like that.

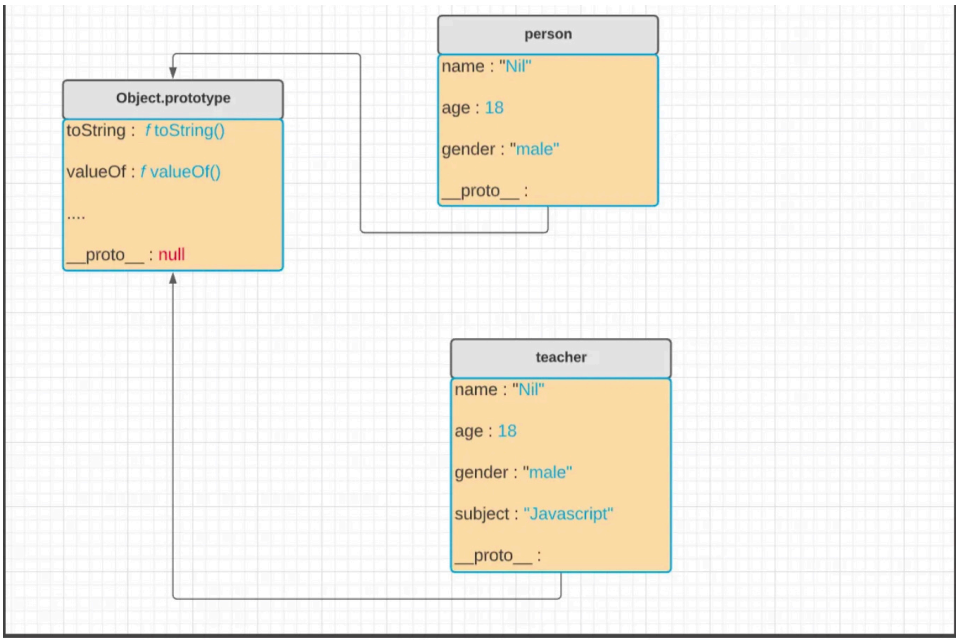



Person.__Proto__ = Object.prototype
(true)

→ here I am going to create teacher object.

```
let teacher = {
    name: 'nil',
    age: 18
    gender : 'male'
    subject: 'Javascript'
}
```

teacher.__proto__ = object.prototype
(true)



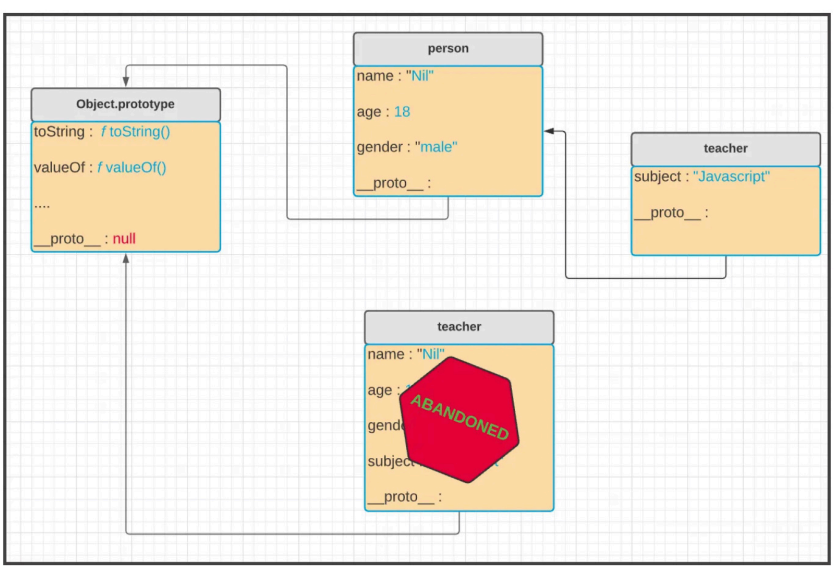

→ here instead of creating new teacher object I am going to create teacher object with only single property subject and we will inherit other property from person.

DevTools - chrome://new-tab-page/

Elements Console

top Filter A

```
> let teacher = {
      subject : "Javascript"
  }
undefined
> Object.setPrototypeOf(teacher, person)
▶ {subject: "Javascript"}
>
```

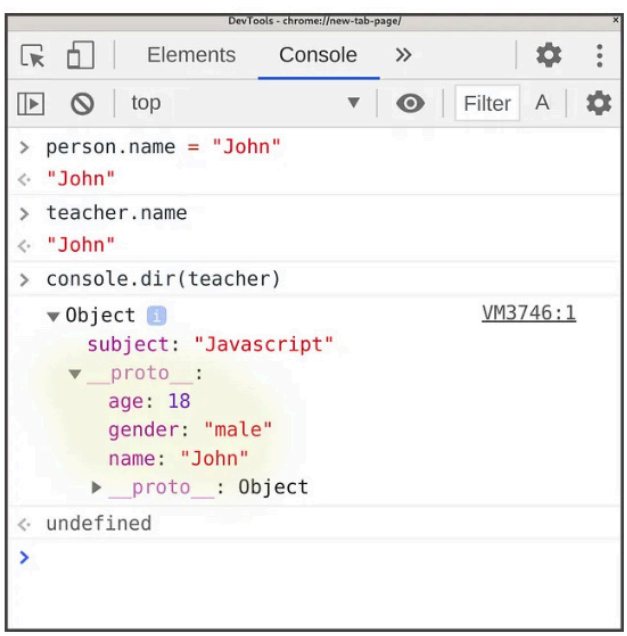



→ teacher's --proto-- property points to the person objects and person's --proto-- property points to Object.prototype object.

teacher.__proto__ = person

(true)

→ one important thing is that __proto__ is just a reference to the prototype. if you modify any property in the prototype object, it would affect the child object as well

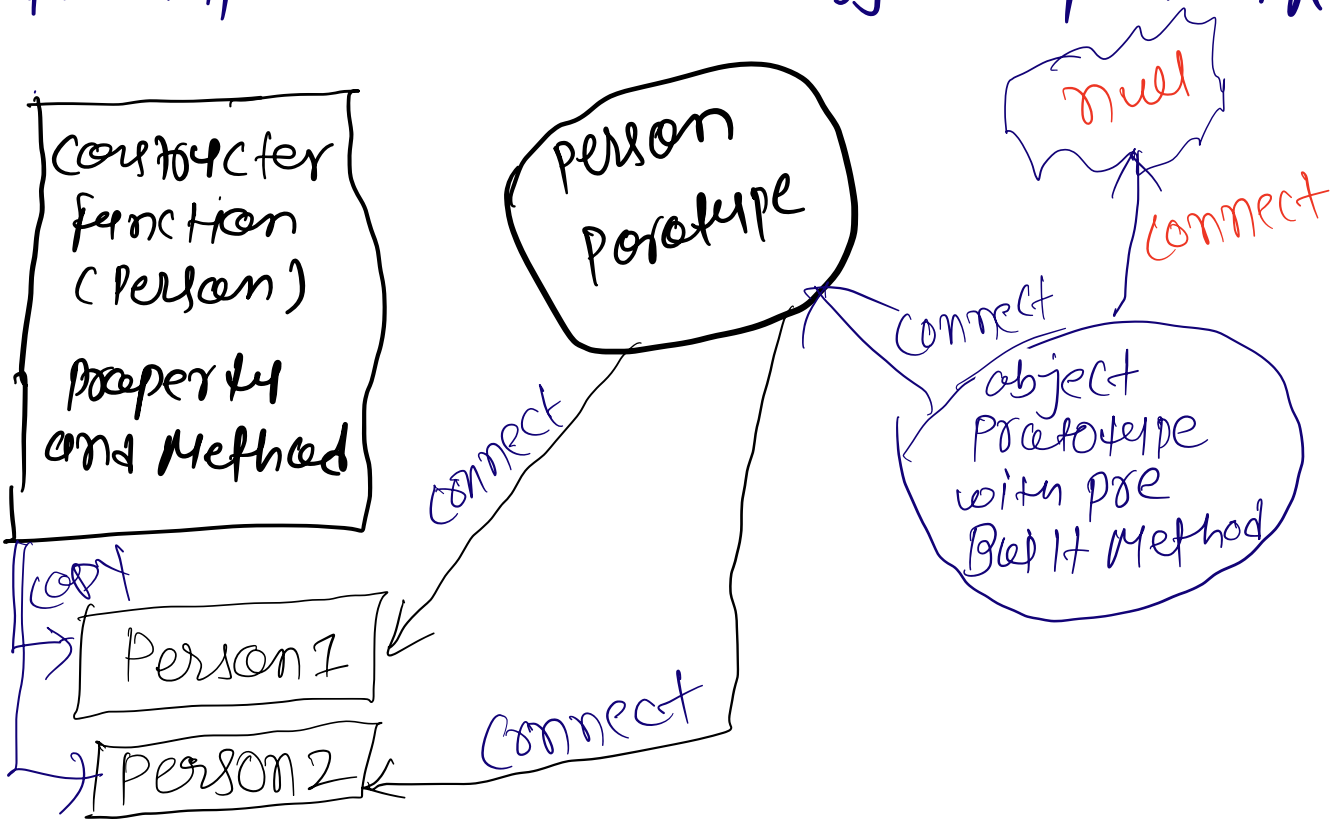



It could be safely said that the term prototypal inheritance is

not entirely accurate.

# Prototype property in constructor function

→ when you are creating object by using constructer function on that time constructor function create own prototype that prototype have another object prototype.



# PROTOTYPE CHAINING

→ prototype object have a one own prototype object which again has a prototype of object until null. This concept of linking prototype object to the parent constructors prototypes all the way up until null is called as Prototype chaining.

```
function Person(){}
Person.prototype.name = "John";
Person.prototype.age = 23;

let john = new Person();

console.log(john); // {}
console.log(john.name);  // 'John
console.log(john.age);  // 23
```

# CREATEING PROTOTYPE

```
function myobject(a, b)
{
    this.a = a
    this.b = b
}
myobject.prototype.mymethod =
    function() {
        console.log(this.a + this.b)
    }
var obj = new myObject('x', 'y')
var obj1 = new myobject('2', '4')

obj1.mymethod()
obj2.mymethod()
```

# Override existing Method

```
console.log(S2.toString())
// [object, object]

Object.prototype.toString = function(){
return "This is custom msg";
}
console.log(S2.toString())
```

Note → overriding built in method on global prototypes should be done with caution, as it can effect the behavior of all object of that type throughout your application.

# PROTOTYPE INHERITANCE

```
// Define a Parent constructer function
function Parent() {
this.parentProperty = 'I am a parent';
// Add a method to the parent's prototype
Parents.prototype.parentMethod = function() {
console.log('I am parent method');
// Define a child constructed function
function Child() {
this.childProperty = 'I am child';
// Inherit from the Parent's prototype
Child.prototype = new Parent()
// Add a method to child' prototype
Child.prototype.childMethod = function() {
console.log('I am child method');
var childInstance = new Child();
```

c.log ( childInstance.parentProperty )
childInstance.parentMethod ()
log (childInstance.childProperty )
childInstance.childMethod ()

```
/ Define a prototype object
onst animalPrototype = {
 speak: function() {
   console.log(`I am ${this.name}`);
 },
;

/ Create an object that inherits from th
onst genericAnimal = Object.create(anima

enericAnimal.name = "Generic Animal";


/ Define a new object that inherits from
onst dog = Object.create(animalPrototype
og.name = "Fido";
/ Add a method specific to the dog objec
og.bark = function() {
 console.log(`${this.name} is barking!`)
;


enericAnimal.speak(); // Output: I am Ge
og.speak(); // Output: I am Fido
og.bark(); // Output: Fido is barking!
```

c.log ( childInstance.parentProperty )
childInstance.parentMethod ()
log (childInstance.childProperty )
childInstance.childMethod ()

Creating a method in the prototype rather than the object in JavaScript has the advantage of memory efficiency. When you attach a method to an object's prototype, it is shared among all instances of that object. This means that the method is stored in memory only once, regardless of how many instances you create.

On the other hand, if you add a method directly to each object instance, it consumes more memory as each instance carries its own copy of the method.