"oop in js"

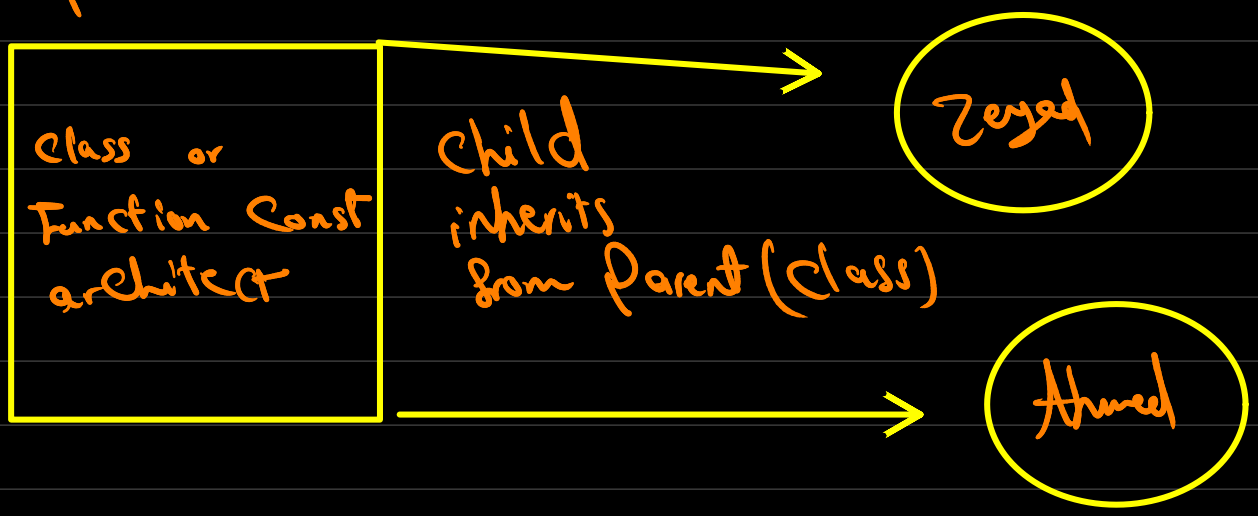
```
const Person = function (firstName, birthYear) {
  this.birthYear = birthYear;
  this.firstName = firstName;
};

const zeyad = new Person('zeyad', 2003);
console.log(zeyad);
```

* Person is called the Constructor Function which is the blue built of ahouse and other lang like C++, ..., it is called the class.

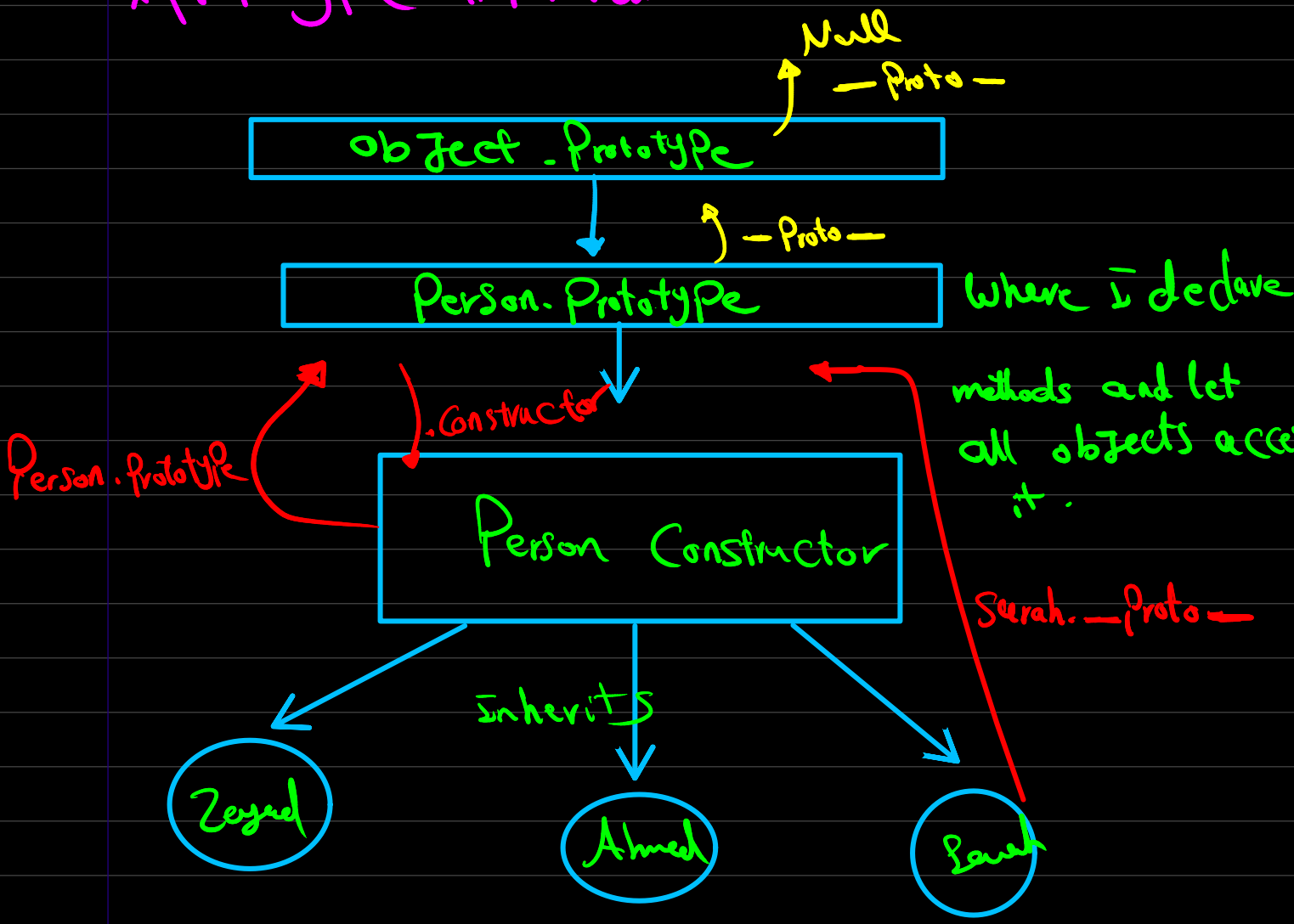
* zeyad is called an object inherited from the person function Constructor.

* Imagine it like this: →



* Zeyad and ahmed inherits From Func Constructor.

*Prototype inheritance :→



note that:— Person.Prototype === Sarah.—Proto—

adding methods to the Prototype

```
Person.prototype.calcAge = function () {
  console.log(2037 - this.birthYear);
};
```

*Es6 Classes :→

It is similar to the prev method but the methods are declared inside the class after constructor but it will be in the prototype not the object itself. (Follow the prev page for more info)

```
// class declaration
class PersonCl {
  constructor(firstName, birthYear) {
    this.firstName = firstName;
    this.birthYear = birthYear;
  }

  calcAge() {
    console.log(2037 - this.birthYear);
  }
}

const jessica = new PersonCl('Jessica', 1996);
console.log(jessica);
jessica.calcAge();

console.log(jessica.__proto__);
```

* So that it doesn't affect the performance, It looks like that.

```
▼ PersonCl {firstName: 'jessica', birthYear: 1996} i
    birthYear: 1996
    firstName: "jessica"
  ▼ [[Prototype]]: Object
    ▶ calcAge: ƒ calcAge()
    ▶ constructor: class PersonCl
    ▶ [[Prototype]]: Object
```

note that the CalcAge exists in the Prototype automatically without the need to do

Person.Prototype.calcAge = function() {};

↳ implementation in the prev way.

*Getter and setters :→

* like other property not treated like methods that require Calling.

* to make it getter or setter we define a keyword of get,set.

```
const account = {
  owner: 'Jonas',
  movements: [200, 530, 120, 300],

  get latest() {
    return this.movements.slice(-1).pop();
  },

  set latest(mov) {
    this.movements.push(mov);
  },
};

console.log(account.latest);

account.latest(50) ;
```

→ not methods

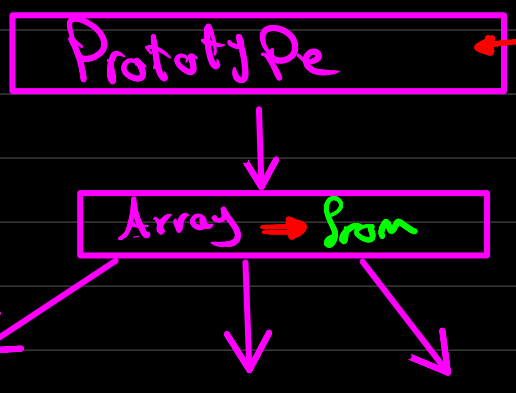
instead → account.latest=50;

Static methods :→

There are methods that are attached to the Constructor itself no other children can access it, as it doesn't exist in the Prototype.

Array.from( );
Number.parseFloat( );

→ Both of them are attached to the Constructor it self.



Push exists in the Prototype so that all children can access it.

nums.push( ) ✓

nums.from( ) ✗

It exists in the Constructor it self so it is not allowed for children to access it.

Static methods in ES6 Classes:→

It is simply by static keyword before its method name.

```
static sayHey() {
  console.log('hey');
}
```

* Static methods in function Constructor way:→

ConstructorName.methodName = function() {};

↳ without mentioning .__proto__

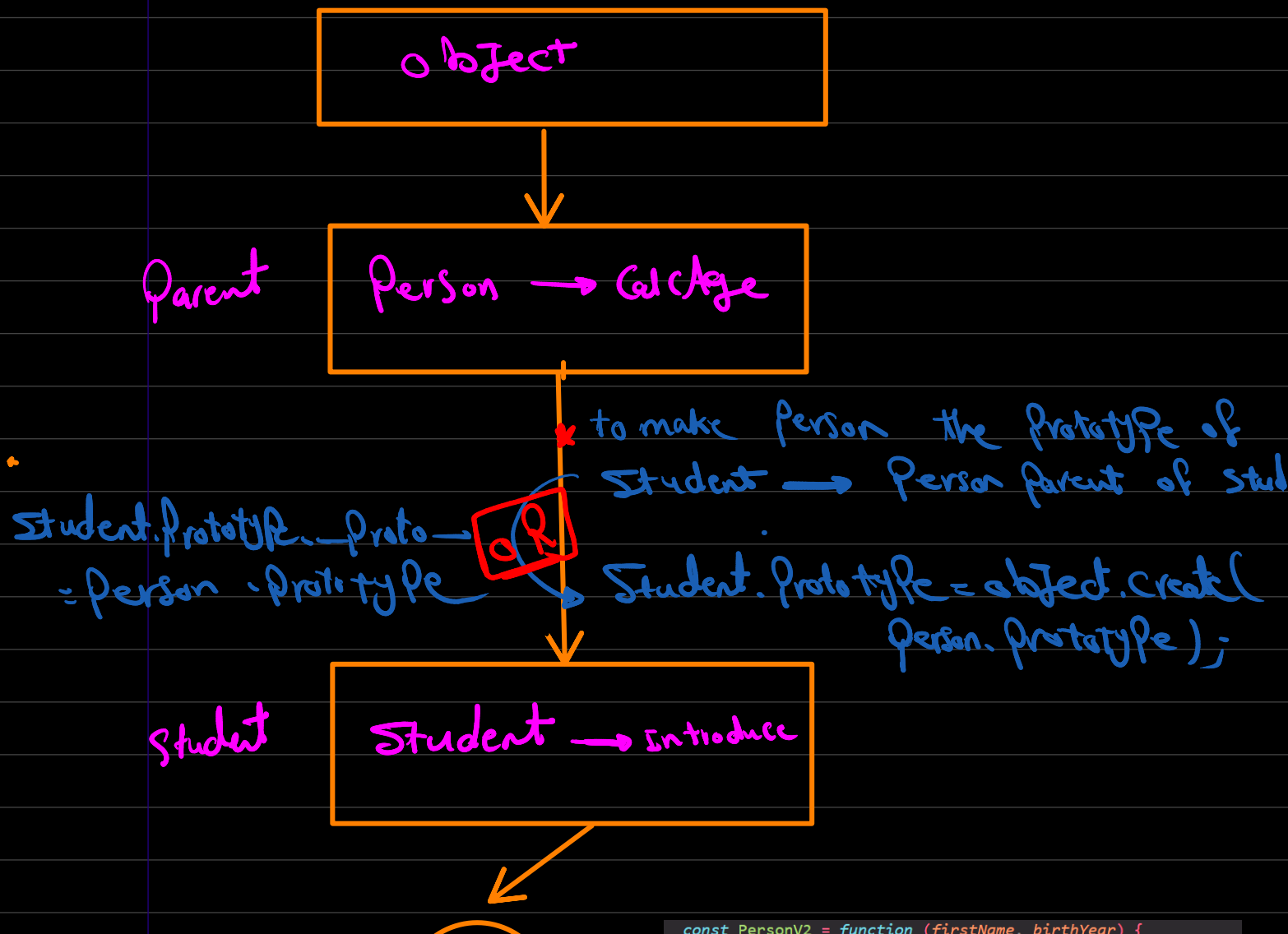
Object.create:→

if I wanna specify the Prototype manually

```
const PersonProto = {
  calcAge() {
    console.log(2037 - this.birthYear);
  },
};

const steven = Object.create(PersonProto);
console.log(steven);
steven.name = 'Steven';
steven.birthYear = 2002;
steven.calcAge();
```

Classes Inherit another Class



*By this way Zeyad can access the methods of his parent which is Student or his grandpa which is Person.

```
const PersonV2 = function (firstName, birthYear) {
  this.firstName = firstName;
  this.birthYear = birthYear;
};

PersonV2.prototype.calcAge = function () {
  return 2024 - this.birthYear;
};
const Student = function (firstName, birthYear, course) {
  PersonV2.call(this, firstName, birthYear);
  this.course = course;
};

// Link Student prototype with the personV2 prototype
// Student.prototype.__proto__ = PersonV2.prototype;
// OR
Student.prototype = Object.create(PersonV2.prototype);
Student.prototype.constructor = Student;
const zeyadV2 = new Student('Zeyad', 2003, 'ECE Dept');
Student.prototype.introduce = function () {
  console.log(`Hey EveryOne!, I'm ${this.firstName} and
  study ${this.course}`);
};
```

Regular Func (Call)

* Inheritance Between classes ES6 :→

* child extends Parent
* Super will Call the Parent automatic and set the this keyword with mentioned params.

```
20  class StudentCl extends PersonCl {
21    constructor(fullName, birthYear, course) {
22      // Always needs to happen first!
23      super(fullName, birthYear);
24      this.course = course;
25    }
26
27    introduce() {
28      console.log(`My name is ${this.fullName} and I
        study ${this.course}`);
29    }
30  }
31
```

* if I specify a function with the same name of the parent, the child func will override the one of the parent.

* Inheritance Between Classes → object. Create :→

```
51  const PersonProto = {
52    calcAge() {
53      console.log(2037 - this.birthYear);
54    },
55
56    init(firstName, birthYear) {
57      this.firstName = firstName;
58      this.birthYear = birthYear;
59    },
60  };
61
62  const steven = Object.create(PersonProto);
63
64  const StudentProto = Object.create(PersonProto);
65  StudentProto.init = function(firstName, birthYear,
    course) {
66    PersonProto.init.call(this, firstName, birthYear);
67    this.course = course;
68  }
69
70  const jay = Object.create(StudentProto);
71  jay.init('Jay')
```

*Private class fields and methods :—→

Private fields are the properties which is Private means that this property will be in the class itself and no one out of the class can get

*It is defined by # → makes Private