4

4 Objectes.

En aquesta lliçó veurem els objectes i com, amb javascript, es poden implementar els conceptes principals de la Programació Orientada a l'Objecte. Utilitzarem l'especificació ES6 que és l'actual dels navegadors moderns.

4.1 Objectes.

El llenguatge Javascript es basa en un model simple d'objectes. En javascript, tot, excepte els valors primitius, és una mena d'objecte, fins i tot, les funcions.

Un objecte és una col·lecció de propietats i mètodes. Les propietats són, en realitat, variables Javascript que pertanyen a l'objecte i defineixen el seu estat. Una propietat pot ser, alhora, un altre objecte. Els mètodes són les funcions associades als objectes i defineixen el seu comportament.

Javascript té un gran nombre d'objectes predefinits: `Number`, `String`, `Date`, etc. També, quan hem carregat una pàgina web al navegador, aquest ha elaborat el DOM (Model d'Objectes del Document), i per manipular-lo des de Javascript cal conèixer com usar objectes per tal de consultar o modificar les seves propietats així com executar els seus mètodes (funcions pròpies de l'objecte).

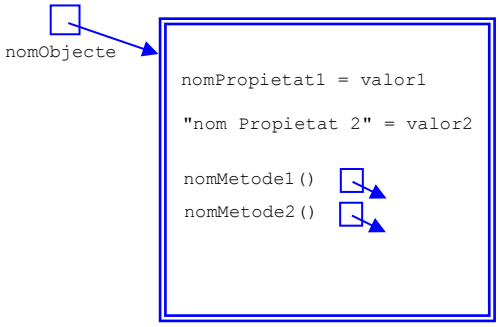
4.1.1 Creació d'objectes. Propietats i mètodes

Un objecte es pot crear de tres formes diferents:

1. **ES3, Usant una constant literal,**

   La sintaxi bàsica és:

```
var nomObjecte = {
  nomPropietat1    :valor1,
  "nom Propietat 2":valor2,
  nomMetode1: function(){},
  nomMetode2: function(){}
};
```



on la variable objecte, `nomObjecte`, conté l'adreça de la memòria de l'ordinador on s'emmagatzema les propietats i mètodes de l'objecte (es parla de que la variable és una referència).

Les **propietats** són variables internes a l'objecte. Per això es parla de que l'objecte és un conjunt de dades en forma de "llista associativa". Els `nomPropietatX` han de ser identificadors javascript vàlids o *strings*, `"nom Propietat X"` on, en aquest cas, podem tenir caràcters no permesos com a identificador, per exemple, els espais en blanc. Els `valorX` són expressions que avaluen un tipus primitiu o un altre objecte. El conjunt de valors que prenen les propietats de l'objecte és l'**estat de l'objecte**.

Els **mètodes** són funcions pròpies de l'objecte destinades a canviar o consultat els valors de les propietats de l'objecte (el seu estat), es diu que defineixen el comportament de l'objecte.

Podem accedir a una propietat concreta de l'objecte amb la notació punt,

```
nomObjecte.nomPropietat1
```

o amb la notació array,

```
nomObjecte["nomPropietat1"]
```

La notació array es fa servir quan es vol accedir a una propietat a través del valor d'una variable. Per exemple,

```
var propietat="nomPropietat1";
nomObjecte[propietat]=3.2;
```

i també quan el nom de la propietat és una cadena de caràcters (String) que conté espais en blanc. Per exemple,

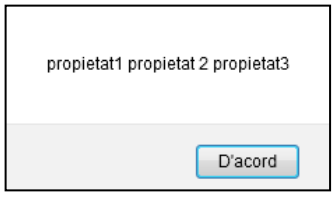
```
var obj={"primera propietat":"hola", "propietat 2":3.2, tercera:15}
obj["primera propietat"]="que tal"; // ara la propietat té el valor "que tal"
obj["propietat 2"]=5.6;
obj["tercera"]=22;
obj.tercera=43; // aquesta propietat també es pot accedir amb la notació punt
```

Hi ha una sentència `for..in..` especial per executar un conjunt de sentències per a tota propietat d'un objecte qualsevol,

```
for(var propietat in nomObjecte){ // es llegeix, per tota propietat de nomObjecte ...
   // accedim als valors de les propietats amb nomObjecte[propietat]
}
```

Per exemple,

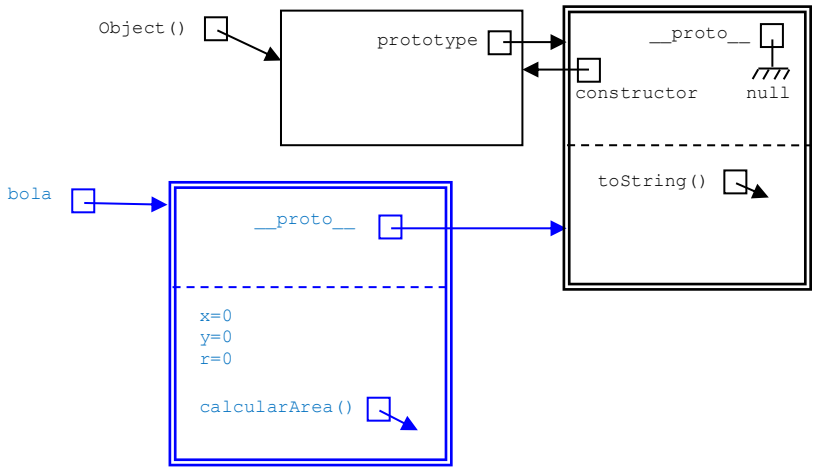
```
var obj={propietat1:"hola", "propietat 2":3.2, propietat3: 24}

var x="";
for(var propietat in obj)
  x=x+propietat+" ";  // escriu tots els noms de les propietats
alert(x);
```




```
for(var propietat in obj)
  x=x+obj[propietat]+" ";  // escriu tots els valors de les propietats.
alert(x);                  // NO es pot utilitzar la notació punt perquè
                           // és una variable
```

Tots els objectes hereten directa o indirectament de l'objecte `Object`, i per tant, es poden fer servir els mètodes de `Object`. Ho veiem amb un exemple,

```
var bola = {
// propietats
  x: 0, y: 0,       // posició
  r: 0,             // radi
  calcularArea: function(){ // mètode
    return Math.PI*this.r*this.r;  // podem actualitzar la propietat r de l'objecte a través de this
  }
};
```




```
bola.r=4 ;         // es pot actualitzar qualsevol propietat. No existeix la privacitat.
bola.color="#ffb"; // podem afegir propietats en temps d'execució
delete bola.color;  // i també, eliminar-les
console.log(bola.calcularArea());   //--> 50.26548,  podem utilitzar mètodes propis on, this==bola
console.log(bola.toString()); //--> "[object Object]" podem utilitzar mètodes "heretats" de Object.
                              // Si no l'hem sobreescrit, s'usa el d'Object que indica només el
                              // tipus o classe d'objecte
```

Tot objecte té una propietat `__proto__` , que no es pot manipular directament, "apuntant" (és una referència) a l'objecte del qual "hereta" propietats i mètodes. Si no s'indica de quin objecte s'està heretant, s'assumeix que s'hereta de `Object`, per tant, tot objecte, directa o indirectament hereta de `Object`.

Quan es demana a un objecte una propietat o mètode que no és seu, es segueix la cadena de `__proto__` per tal de trobar-lo en algun dels objectes de la cadena de `__proto__`. Acaba la cerca quan troba `null`.

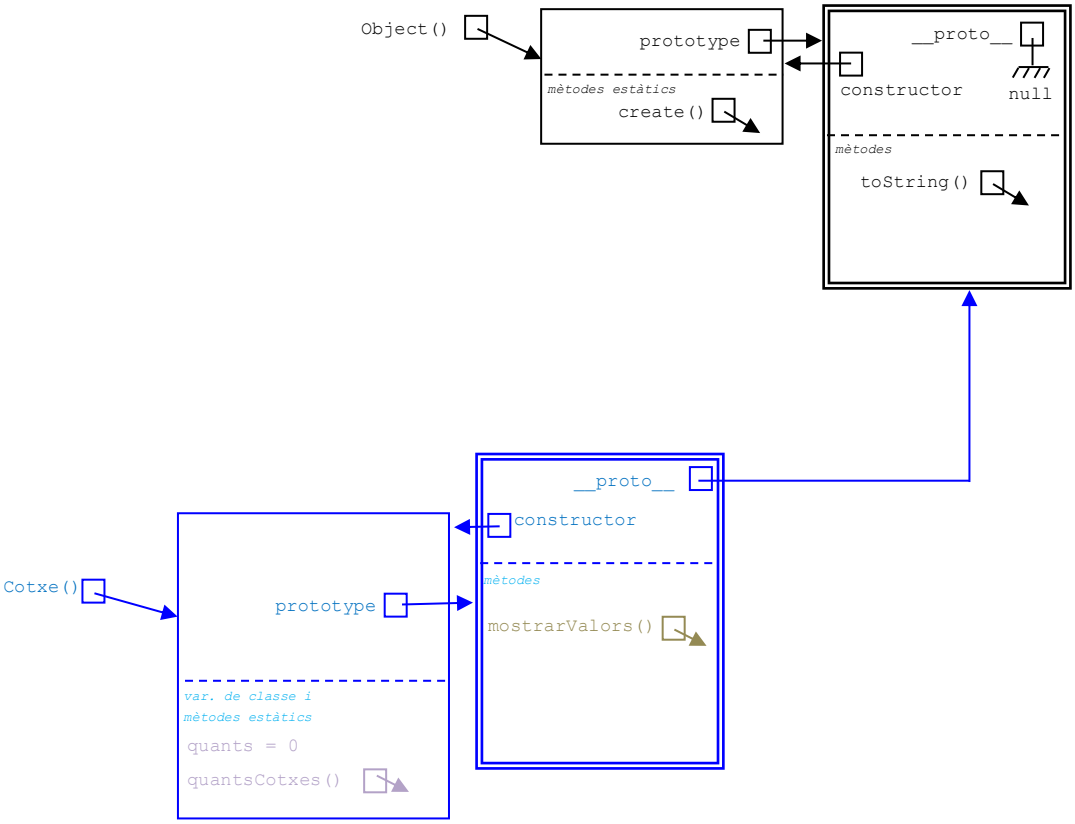
Aquest mètode de crear objectes es fa servir quan només hi ha una instància d'aquest objecte en el programa o quan l'objecte només té propietats.

2. ES3, usant la funció constructora d'objectes,

Tota funció javascript és potencialment creadora d'altres objectes. Per diferenciar aquestes funcions creadores de les altres, se solen identificar amb una primera lletra majúscula i es parla de que construeix la "classe" d'objectes corresponent.

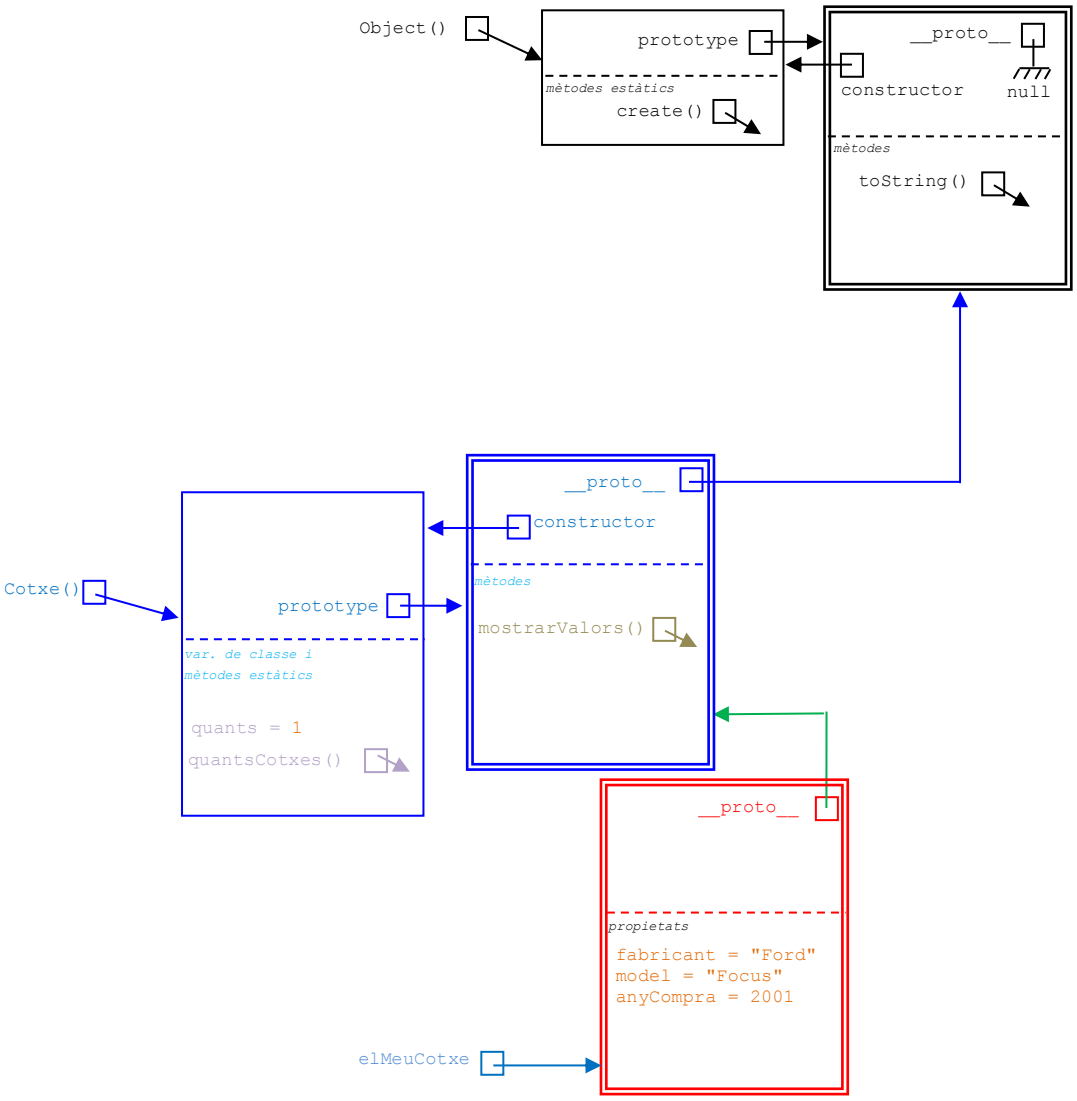
Per exemple,

```
function Cotxe(fabricant,model,anyCompra){
  // propietats
  this.fabricant=fabricant;
  this.model=model;
  this["any de compra"]=anyCompra;
  Cotxe.quants++;
}
// mètodes
Cotxe.prototype.mostrarValors= function(){
  alert(this.fabricant+" "+this.model+" "+this["any de compra"]);
};
// variables de classe i mètodes estàtics
Cotxe.quants=0;
Cotxe.quantsCotxes=function(){return Cotxe.quants;}
```



i ara, executem la següent instrucció, per crear l'objecte:

```
var elMeuCotxe=new Cotxe("Ford","Focus",2001);

console.log(Cotxe.quantsCotxes()); // --> 1
```



Amb aquesta segona manera de "construir" objectes és més fàcil crear-los, només cal executar **`new`** amb la funció constructora de l'objecte passant-li els valors inicials de les propietats de l'objecte (estat inicial). Aquest operador **`new`** crea l'objecte, executa la funció constructora on **`this`** fa referència a l'adreça de l'objecte que s'està construint.

Fixeu-vos que els mètodes estan en l'objecte `prototype` de la funció constructora i, per tant, són accessibles des de qualsevol objecte creat amb aquesta funció constructora. De fet, si fem `elMeuCotxe.mostrarValors()` es busca el mètodo en el propi objecte `elMeuCotxe`, i com que no el troba, segueix per la cadena de `__proto__` fins arribar a trobar el mètode o trobar **`null`**.
La conclusió més important és que cada objecte té les seves pròpies propietats (estat) però, comparteixen els mètodes de la funció constructora (comportament)

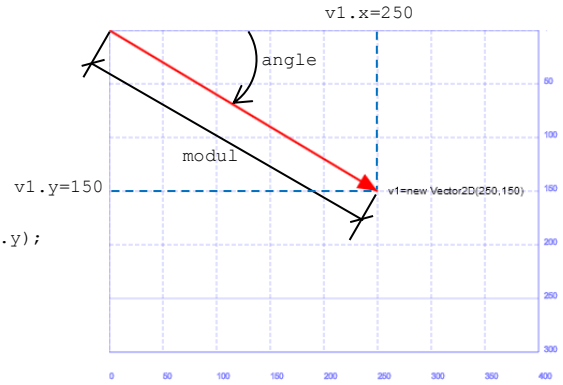
Nota 1: la propietat `__proto__` que té tot objecte només es pot manipular a través de l'operador **`new`**.

3. ES6, Definint una classe d'objectes.

En l'especificació EMACScript 6 (any 2015), es poden declarar classes d'objectes, tot i que la implementació segueix sent amb prototipus. Han aparegut les paraules claus noves: **class**, **constructor**, **extends**, **super**, **static**.
Per exemple (provat en el Firefox 46.0, i el Chrome 50.0)

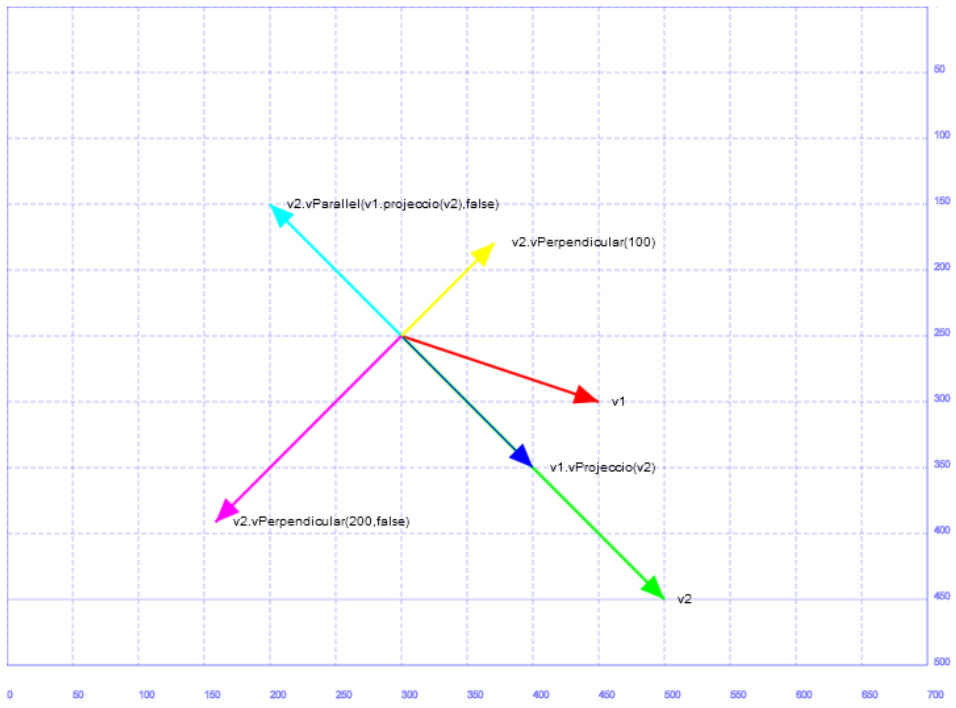
```
class Punt{                 // definim la classe d'objecte Punt
    constructor(x,y){       // funció constructora de la classe
        this.x=x;
        this.y=y;
    }
    static distancia(a,b){  // mètode estàtic o de classe; no cal posar function
        const dx=a.x-b.x;
        const dy=a.y-b.y;
        return Math.sqrt(dx*dx+dy*dy);
    }
}

class Forma{
    constructor(posicio){
        this.posicio=posicio;   // aquesta propietat és un Punt --> composició d'objectes
    }
    get area(){}  // no està definit, mètode abstracte
}

class Rectangle extends Forma{              // herència de classes
  constructor(posicio, amplada, alçada){
     super(posicio);                        // cridem al constructor base
     this.amplada=amplada;
     this.alçada=alçada;
  }
  get area(){  // definim la propietat area
    return this.amplada* this.alçada;
  }
}


const p1=new Punt(5,5);
const p2=new Punt(10,10);
console.log("distància entre els dos punts és "+Punt.distancia(p1,p2));
// --> distància entre els dos punts és 7.0710678118654755

var r=new Rectangle(new Punt(0,0), 10, 50);
console.log("l'àrea del rectangle és " + r.area);

// --> l'àrea del rectangle és 500
console.log("la posició del rectangle és: x = " + r.posicio.x + " , y = " + r.posicio.y);

// --> la posició del rectangle és: x = 0 , y = 0
```

4.2 Exemple: Vector2D.




```
class Vevtor2D{
  constructor(x,y) {
    this.x = x;
    this.y = y;
  }

  // MÈTODES PÚBLICS

    modul(){
        return Math.sqrt(this.x*this.x + this.y*this.y);
    }
    angle(){
        return Math.atan2(this.y,this.x);
    }
    vClone() {
        return new Vector2D(this.x,this.y);
    }
    vUnitari() {
        var modul = this.modul();
        if (modul > 0) {
            return new Vector2D(this.x/modul,this.y/modul);
        }else{
            return new Vector2D(0,0);
        }
    }
    normalitzat() {
        var modul = this.modul();
        if (modul > 0) {
            this.x /= modul; this.y /= modul;
        }
        return this.modul();
    }
    vSumar(vec) {
        return new Vector2D(this.x + vec.x,this.y + vec.y);
    }
    sumar(vec) {
        this.x += vec.x; this.y += vec.y;
    }
    vRestar(vec) {
        return new Vector2D(this.x - vec.x,this.y - vec.y);
    }
    restar(vec) {
        this.x -= vec.x; this.y -= vec.y;
    }
    vEscalar(k) {
        return new Vector2D(k*this.x,k*this.y);
    }
    vSumarEscalat(vec,k) {
        return new Vector2D(this.x + k*vec.x, this.y + k*vec.y);
    }
    escalar(k) {
        this.x *= k;  this.y *= k;
    }
    producteEscalar(vec) {
        return this.x*vec.x + this.y*vec.y;
    }
    projeccio(vec) {
        var modul = this.modul();
        var modulVec = vec.modul();
        var proj;
        if( (modul == 0) || ( modulVec == 0) ){
            proj = 0;
        }
        else {
            proj = (this.x*vec.x + this.y*vec.y)/modulVec;
        }
        return proj;
    }
    vProjeccio(vec) {
        return vec.vParallel(this.projeccio(vec));
    }
    vParallel(u,positiu){
        if (typeof(positiu)==='undefined') positiu = true;
        var modul = this.modul();
        var vec = new Vector2D(this.x, this.y);
        if (positiy){
            vec.escalar(u/modul);
        }
        else{
            vec.escalar(-u/modul);
        }
        return vec;
    },
    vPerpendicular(u,antihorari){
```

```
        if (typeof(antihorari)==='undefined') antihorari = true;
        var modul = this.modul();
        var vec = new Vector2D(this.y, -this.x);
        if (modul > 0) {
            if (antihorari){ // antihorari respecte el sistema de coordenadas del canvas
                vec.escalar(u/modul);
            }else{
                vec.escalar(-u/modul);
            }
        }else{
            vec = new Vector2D(0,0);
        }
        return vec;
    }


    // MÈTODES ESTÀTICS
    static distancia(vec1,vec2){
        return (vec1.vRestar(vec2)).modul();
    }
    static angleEntre(vec1,vec2){
        return Math.acos(vec1.producteEscalar(vec2)/(vec1.modul()*vec2.modul()));
    }
    static vector2D(modul,angle,horari){
        if (typeof(horari)==='undefined') horari = true;
        var vec = new Vector2D(0,0);
        vec.x = modul*Math.cos(angle);
        vec.y = modul*Math.sin(angle);
        if (!horari){
            vec.y *= -1;
        }
        return vec;
    }
}

var p={x:300,y:250};
var v1=new Vector2D(150,50);
var v2=new Vector2D(200,200);
v1.draw(context,p,"#f00",2,"v1");
v2.draw(context,p,"#0f0",2,"v2");
v1.vProjeccio(v2).draw(context,p,"#00f",1," v1.vProjeccio(v2)");
v2.vParallel(v1.projeccio(v2),false).draw(context,p,"#0ff",2," v2.vParallel(v1.projeccio(v2),false)");
v2.vPerpendicular(100).draw(context,p,"#ff0",2," v2.vPerpendicular(100)");
v2.vPerpendicular(200,false).draw(context,p,"#f0f",2," v2.vPerpendicular(200,false)");
```



4.3 Guardar informació.

Existeixen uns magatzems, en realitat són objectes amb una estructura de llista associativa, que ens permeten emmagatzemar informació de l'aplicació de forma permanent (al disc dur de l'ordinador client) o mentre duri la sessió de l'usuari (no tanqui el navegador). Aquests objectes són: `localStorage` i `sessionStorage`, respectivament.

El `localStorage` serveix per guardar l'estat de l'aplicació. Així, podem tancar l'ordinador i en un futur, tornar a executar l'aplicació en el mateix ordinador recuperant l'estat amb que es fa finalitzar la darrera execució.

El `sessionStorage` serveix per poder passar informació d'una pàgina html a una altra mentre es mantingui la sessió de l'usuari. Una sessió es manté mentre no es tanqui l'execució actual del navegador.

		propietats i mètodes	descripció
`localStorage` `sessionStorage`	.	`length`	retorna la quantitat d'items que hi ha al magatzem.
		`key(index)`	retorna la clau que està a la posició `index`.
		`getItem(clau)`	retorna el valor associat a la clau. Si la clau no hi és, retorna `null`.
		`setItem(clau, valor)`	afegeix la parella clau-valor. Si la clau ja existeix, actualitza el valor.
		`removeItem(clau)`	elimina la parella clau-valor.
		`clear()`	buida el magatzem.

El tipus de les claus i els valors de la llista associativa són `String`.

4.4 JSON

En certes circumstàncies podem necessitar emmagatzemar, de forma permanent, els valors de les nostres variables, per tal de poder-los recuperar més tard, fins i tot en altres aplicacions. Es parla de la serialització de l'entorn. Per fer-ho universal, és a dir, que es pugui recuperar des de qualsevol aplicació, se sol emmagatzamar els valors en format string. L'aplicació que ha de recuperar el valor concret disposarà d'un intèrpret (parser) que el string al valor concret representat.

El javascript disposa d'una notació d'objectes que permet la serialització, és el JSON (JavaScript Object Notation). Bàsicament, s'utilitza la notació de literal del propi javascript, però, amb algunes diferències:

	diferències entre JSON i els literals javascript
objectes i arrays	• els noms de les propietats han de ser `String` amb ", per exemple: `{"edat":54}` • les comes finals estan prohibides, per exemple, és incorrecte: `{"x":2, "y":3.5,}` o `[3,"hola",]`
números	• no poden començar per 0 • si porta el punt decimal, cal almenys, un dígit després del punt
`String`	• sempre s'utilitza les " per emmarcar, no està permés emmarcar amb '

Disposem de dos mètodes estàtics:

- `JSON.parse('stringJSON')` que permeten interpretar un String amb notació JSON i convertir-lo amb una variable javascript. Observeu que el `stringJSON` es passa emmarcat amb ' per no confondre amb les " que pugui contenir el `stringJSON`. Si emmarquen el `stringJSON` amb `"`, caldria posar \ al davant de cada " que contingui el `stringJSON`.
- `JSON.stringify(varJS)` que permet convertir el valor d'una variable a un string amb notació JSON

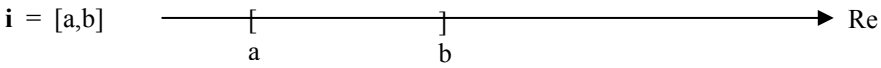
Per exemple,

```
var varJS=JSON.parse('{}');                          // {}
    varJS=JSON.parse('true');                        // true
    varJS=JSON.parse('"hola"');                      // "hola"
    varJS=JSON.parse('[1,.5,"hola"]');               // [1,.5,"hola"]
    varJS=JSON.parse('null');                        // null
    varJS=JSON.parse('{"nom":"Joan", "edat":54}');   // {nom:"Joan", edat:54}
    console.log(varJS.nom);                          // "Joan"

var stringJSON=JSON.stringify({});                  // "{}"
    stringJSON=JSON.stringify(true);                // "true"
    stringJSON=JSON.stringify("hola");              // "\"hola\""
    stringJSON=JSON.stringify([1,"true",true]);     // "[1,\"true\",true]"
    stringJSON=JSON.stringify({x:5});               // "{\"x\":5}"
```

4.5 Exercicis.

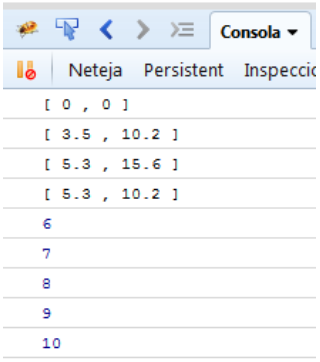
1) Escriviu el codi javascript per a la "classe" d'objectes Interval. Un Interval ve donat per dos nombre reals: a (inici interval) i b (fi interval), ambdós inclosos al interval (també anomenat interval tancat).

   Les operacions que podem fer amb els intervals seran:
   a) la seva construcció,
   b) saber si un nombre real donat pertany al interval,
   c) executar una funció donada per cada enter dins del interval,
   d) saber si dos intervals intersequen o no,
   e) calcular el interval intersecció de dos intervals (part comuna dels dos intervals, si no hi ha intersecció retorna el Interval buit) i
   f) la seva representació com a string.



   Escriviu la classe javascript Interval que permeti executar el següent codi javascript amb el resultat indicat en la imatge:

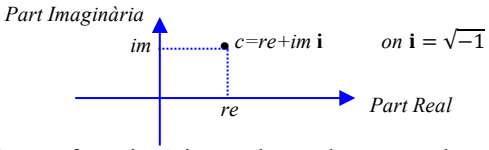
```
console.log(Interval.buit.toString());

var i1=new Interval(3.5,10.2);
var i2=new Interval(5.3,15.6);
var i3;

console.log(i1.toString());
console.log(i2.toString());
if(i1.intersecta(i2))
  i3=Interval.interseccio(i1,i2);
console.log(i3.toString());

if(i3.inclou(7))
  i3.perCadaEnter(console.log);
```



Nota: Observeu que el interval buit el representem com a [0,0] i és un objecte estàtic constant de la classe Interval

```
class Interval{
    constructor(a,b){
     this.a=a;
     this.b=b;
  }
  ////////////// Mètodes públics /////////////////
  inclou(x){
     return this.a<=x && x<=this.b;
  }
  perCadaEnter(f){
     for(var x=Math.ceil(this.a); x<=this.b; x++) f(x);
  }
  intersecta(i2){
    return this.inclou(i2.a) || this.inclou(i2.b) ||
           i2.inclou(this.a) || i2.inclou(this.b);
  }
  toString(){
   return "[ " + this.a + " , " + this.b + " ]";
  }
  ////////////// Mètodes estàtics /////////////////
  static interseccio(i1,i2){
    if(i1.intersecta(i2)){  // si hi ha intersecció
      return new Interval(Math.max(i1.a,i2.a), Math.min(i1.b,i2.b));
     }
     else
       return Interval.buit;
  }
}

////////////// Constants estàtiques /////////////////
Interval.buit=new Interval(0,0);
```

2) Escriviu una "classe" d'objectes Complexe. Recordeu que un complexe és una parella de valors: el primer és la part real i el segon la part imaginària del complexe.



Es pot fer aritmètica amb nombres complexes: sumar, restar, multimplecar, dividir, etc. Per exemple, donats dos complexes: $c_1 = re_1 + im_1\,\mathbf{i}$, $c_2 = re_2 + im_2\,\mathbf{i}$ tenim,

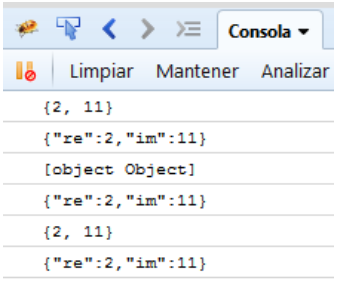
$c_1 + c_2 = (re_1 + re_2) + (im_1 + im_2)\,\mathbf{i}$

$c_1 * c_2 = (re_1 * re_2 - im_1 * im_2) + (re_1 * im_1 + re_2 * im_2)\,\mathbf{i}$

mòdul, $\|c_1\| = \sqrt{re_1^2 + im_1^2}$

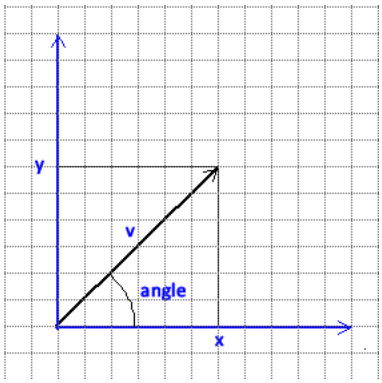
Excriviu una classe Complexe que tingui el següent comportament:

```
var c1=new Complexe(2,2);
var c2=c1.multiplicar(c1.sumar(Complexe.U)).sumar(Complexe.I);
var c3=c2;
console.log(c2.toString());
console.log(JSON.stringify(c2));
localStorage.setItem("c2",JSON.stringify(c2));
c2=JSON.parse(localStorage.getItem("c2"));
console.log(c2.toString());  // ara és un objecte literal
console.log(JSON.stringify(c2));
var c4=new Complexe(c2.re,c2.im);
console.log(c4.toString());  // ara torna a ser un Complexe
if(c3.equals(c4)) console.log(JSON.stringify(c4));
```




```
class Complexe{
    constructor(real,imaginari){  // constructor
        if(isNaN(real) || isNaN(imaginari)) throw new TypeError();
        this.re=real;    this.im=imaginari;
    }
    ////////////// Mètodes públics /////////////////
    sumar(c2){return new Complexe(this.re+c2.re, this.im+c2.im); }
    multiplicar(c2){
        return new Complexe(this.re*c2.re-this.im*c2.im,this.re*c2.im+this.im*c2.re);
    }
    modul(){return Math.sqrt(this.re*this.re+this.im*this.im);}
    toString (){return "{"+this.re+", "+this.im+"}";}
    equals(c2){
        return c2!=null &&
               c2.constructor===Complexe &&
               Math.abs(this.re-c2.re)<0.0000001 && Math.abs(this.im-c2.im)<0.0000001;
    }
    ////////////// Mètodes estàtics /////////////////
}
// constants de classe
Complexe.U = new Complexe(1,0);
Complexe.I = new Complexe(0,1);
```

3) Escriviu el codi javascript per a la "classe" d'objectes vector2D. Un vector2D ve representat per dues coordenades (x,y) o bé pel seu mòdul i angle respecta l'eix X. Els vector2D es poden sumar, restar, escalar (multiplicar per un escalar), el producte escalar de dos vector2D i calcular el vector2D ortogonal:



v = (x,y)

mòdul de **v** = $\sqrt{(x^2 + y^2)}$

angle (radiants) de **v** = arctg(y/x)

$\mathbf{v}_1 + \mathbf{v}_2 = (x_1+x_2, y_1+y_2)$

$\mathbf{v}_1 - \mathbf{v}_2 = (x_1-x_2, y_1-y_2)$

k*$\mathbf{v}$ = (k*x,k*y)

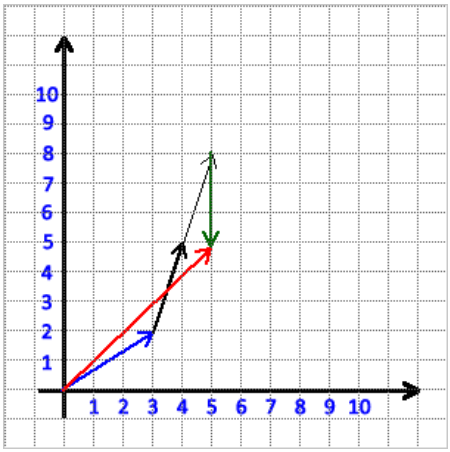
producte escalar, $\mathbf{v}_1 \cdot \mathbf{v}_2 = x_1*x_2+y_1* y_2$

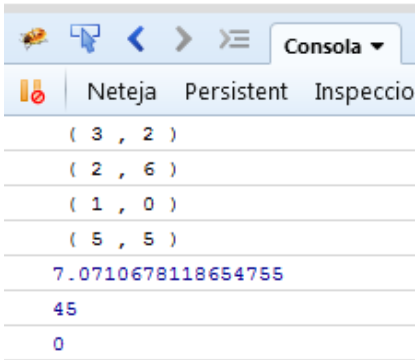
vector ortogonal a **v**, $\mathbf{v}_L$=(-y,x) on $\mathbf{v}_L$ és **v** girat 90º en sentit antihorari

vectors unitaris: **i** = (1,0), **j** = (0,1)

Escriviu la classe javascript Vector2D que permeti executar el següent codi javascript amb el resultat indicat en la imatge:

```
var v1=new Vector2D(3,2);
var v2=new Vector2D(1,3);
var v3;

v3=v1.sumar(v2.perK(2)).restar(Vector2D.i.ortogonal().perK(3));  // v3 = v1 + 2*v2 - 3*iL
console.log(v1.toString());  //  ( 3 , 2 )
console.log(v2.toString());  //
console.log(Vector2D.i.toString());
console.log(v3.toString());
console.log(v3.modul());
console.log(v3.angleGraus());
console.log(Vector2D.producteEscalar(Vector2D.i,Vector2D.j));
```





```
class Vector2D{
    constructor(x,y){
        this.x=x;
        this.y=y;
    }

////////////// Mètodes públics /////////////////
    sumar(v2){
        return new Vector2D(this.x+v2.x, this.y+v2.y);
    }
   restar(v2){
        return new Vector2D(this.x-v2.x, this.y-v2.y);
    }
    perK(k){
        this.x *=k; this.y *=k;
        return this;
    }
   ortogonal(){
        return new Vector2D(-this.y,this.x);
    }
    modul(){
        return Math.sqrt(this.x*this.x + this.y*this.y);
    }
    angleGraus(){
        return Math.atan2(this.y,this.x)*180/Math.PI;
    }
    toString(){
        return " ( "+this.x+" , "+this.y+" ) ";
    }
    ////////////// Mètodes estàtics /////////////////
    static producteEscalar(v1,v2){
      return v1.x*v2.x + v1.y*v2.y;
  }
}



////////////// Constants estàtiques /////////////////
Vector2D.i=new Vector2D(1,0);
Vector2D.j=new Vector2D(0,1);
```