Guia d'ús de programació: logotip per a la revista en línia *Mosaic*

Universitat Oberta de Catalunya

Índex

Autoria: Marc Padró

PID_00267115

Primera edició: setembre 2019
Autoria: Marc Padró

Av. Tibidabo, 39-43, 08035 Barcelona
Realització editorial: FUOC

Guia d'ús de programació: logotip per a la revista en línia *Mosaic*

1. Estructura del projecte

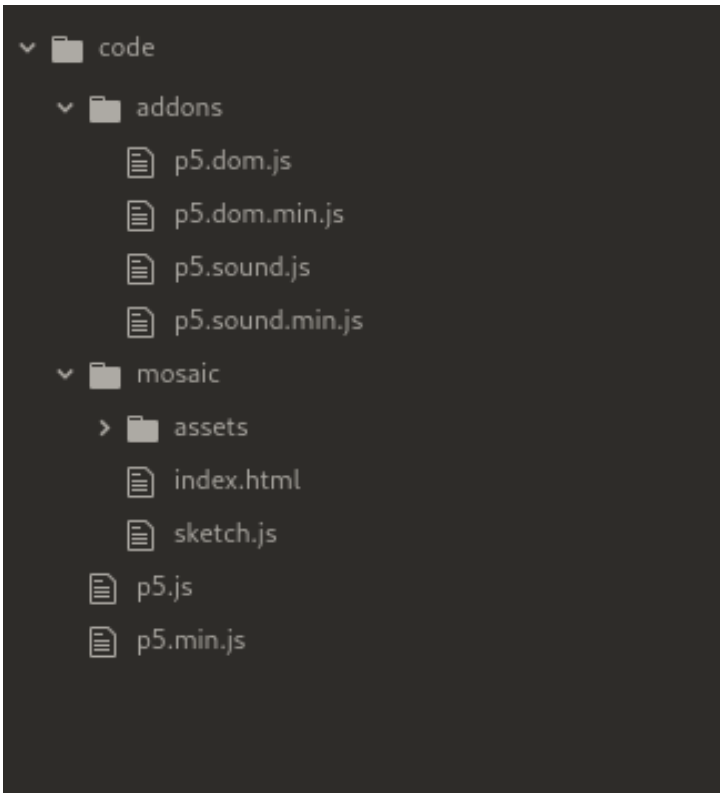
Un projecte de p5.js, anomenat *sketch*, consta principalment dels fitxers següents:

- **index.html:** Conté l'estructura bàsica per accedir al nostre *sketch* mitjançant un navegador web. En aquest document, s'hi enllacen la resta de fitxers del projecte; l'haurem de modificar si canviem la disposició o el nom d'algun d'aquests fitxers o si n'afegim de nous, però a part d'això no caldrà que l'editem. Podem visualitzar el resultat del projecte obrint aquest arxiu amb el nostre navegador.

- **p5.min.js o p5.js:** Conté el codi base de p5.js que necessitarem per fer funcionar qualsevol *sketch*. El llenguatge de base del *sketch* és JavaScript, però necessitem aquest arxiu per afegir tota la funcionalitat extra que ens permet escriure codi de p5.js. És el que s'anomena una *llibreria*.

- **sketch.js:** És on escrivim el nostre codi.

Figura 1. Arxius i estructura del projecte



A més, en aquest cas, al directori 'assets' s'hi troben tots els recursos que carreguem en el nostre programa:

- **Imatges per als números**, de 0 a 9. Per a cada número hi ha una versió normal i una de petita. Com que són imatges petites i només tenen un color i el fons transparent, les hem guardat en format GIF, que és el que pesava menys. Hem fet servir el color blanc, perquè d'aquesta manera després les podem mostrar amb el color que vulguem fent servir la funció 'tint'.

- **Imatge de fons**. Fem servir una imatge per al fons, ja que no haver-lo de dibuixar des del codi ens estalvia molta feina. La imatge té el doble de resolució de la mida a la qual es mostrarà per tal que es vegi millor en pantalles d'alta densitat (*retina* i equivalents). No fem servir un SVG perquè p5.js el llegiria com un mapa de bits igualment i pot ser que en alguns casos no acabi de funcionar correctament.

- **Tipografia** que farem servir per al text, en format TTF. Tot i que en la versió definitiva farem servir una imatge, aquí utilitzarem text perquè sigui més flexible. Com que no podem incloure la tipografia original, hem fet servir una alternativa.

Figura 2. Contingut del directori 'assets'

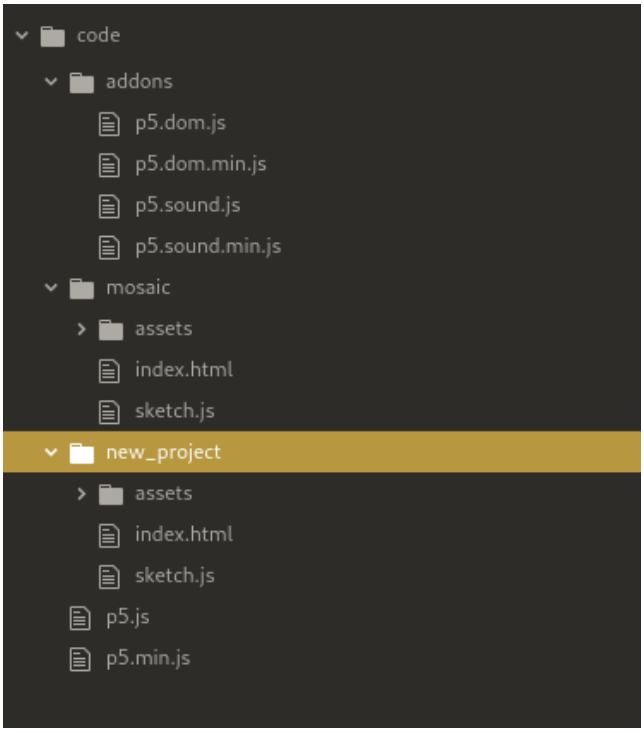
Com que l'objectiu serà integrar l'*sketch* en una plataforma Wordpress mitjançant l'extensió 'p5.js Embed' (en podeu consultar el funcionament a la guia d'ús corresponent), al final només necessitarem el contingut de l'arxiu 'sketch.js' i el directori 'assets'. Però mentre no fem aquesta integració podem treballar en el nostre *sketch* de manera aïllada; llavors necessitarem també la resta de fitxers.

2. Com treballar en el nostre propi projecte

2.1. Crear el projecte

Per començar a treballar en el vostre propi projecte, només cal que creeu una nova carpeta dins el directori 'code' i que hi copieu els continguts de la carpeta 'mosaic'. D'aquesta manera, podem compartir la llibreria de p5.js entre tots els projectes que creem sense haver-la de duplicar. Si voleu començar des de zero, podeu eliminar el contingut de l'arxiu 'sketch.js' i tots els documents del directori 'assets'.

Figura 3. Podem crear un nou projecte copiant els continguts de la carpeta 'mosaic'



Si volem fer servir algun *addon* de p5.js (http://p5js.org/libraries/) necessitarem enllaçar l'arxiu corresponent des d''index.html'. Els arxius per a les llibreries p5.dom i p5.sound es troben ja al directori 'addons', però haurem de modificar lleugerament l'arxiu 'index.html' perquè carregui les llibreries que vulguem.

Si obriu 'index.html' en un editor, veureu que hi ha dues línies comentades, és a dir, entre els caràcters "<!--" i "-->".

Figura 4. Les línies 9 i 10 de l'arxiu index.html estan comentades

```
7   <style> body {padding: 0; margin: 0;} </style>
8   <script src="../p5.min.js"></script>
9   <!-- <script src="../addons/p5.dom.min.js"></script> -->
10  <!-- <script src="../addons/p5.sound.min.js"></script> -->
11  <script src="sketch.js"></script>
12 </head>
13 <body>
```

Podeu eliminar aquests caràcters per enllaçar la llibreria corresponent.

Figura 5. Eliminem els marcadors de comentari del la línia 10 per enllaçar la llibreria p5.sound

```
7   <style> body {padding: 0; margin: 0;} </style>
8   <script src="../p5.min.js"></script>
9   <!-- <script src="../addons/p5.dom.min.js"></script> -->
10  <script src="../addons/p5.sound.min.js"></script>
11  <script src="sketch.js"></script>
12 </head>
13 <body>
```

Possiblement us deveu haver adonat que tant la llibreria principal com els 'addons' s'enllacen mitjançant un arxiu el nom del qual conté el text "[...].min[...]"; però que per a cada llibreria hi ha també un arxiu sense aquests caràcters. Els arxius que contenen "min" en el seu nom són arxius "minificats", això vol dir que el codi d'aquests arxius s'ha modificat de forma automàtica per tal que ocupi el mínim espai possible, però s'executi de la mateixa manera. Així els fitxers pesen menys i el navegador els pot carregar més ràpidament. El desavantatge d'usar arxius "minificats" és que no es pot entendre el codi i en cas de produir-se un error pot ser més difícil saber-ne el motiu.

En aquest cas, els arxius sense minificar no són necessaris per al funcionament dels nostres programes, però els hem inclòs per si algú té ganes de veure com estan escrites aquestes llibreries. De totes maneres, tampoc no són els arxius originals en què es programa la llibreria p5.js, sinó que es fa en diferents arxius que separen les funcionalitats de la llibreria per mòduls. Podeu consultar els arxius originals de desenvolupament en aquest enllaç: https://github.com/processing/p5.js/tree/master/src.

2.2. Separar el codi

Generalment, si hem de treballar en un projecte complex, ens interessa separar el codi en diversos documents per per tenir-lo més estructurat i ordenat. En aquest exemple, però, l'hem mantingut tot junt, ja que a l'hora d'integrar-lo a Wordpress fent servir l'extensió 'p5.js Embed' necessitem tenir-lo tot junt igualment.

De totes maneres, hem separat les diferents seccions que hauríem posat en fitxers diferents mitjançant el següent estil de comentari:

```
/*
=====================================================================
    NOM SECCIÓ
=====================================================================
*/
```

Trobareu seccions principals per als àmbits següents:

- Variables de configuració
- Programa principal
- Funcions de dibuix
- Classe 'Sector' (Utilitat per definir els quatre sectors del logotip)
- Classe 'Timer' (Utilitat per definir contadors de temps)

Si vosaltres voleu separar el vostre codi en diferents fitxers, només cal que creeu fitxers JS addicionals al mateix directori que 'sketch.js' i els enllaceu des d''index.html'. Per fer-ho podeu duplicar la línia següent:

```
<script src="sketch.js"></script>
```

I substituir 'sketch.js' pel nom del nou document. Per exemple:

```
<script src="sketch.js"></script>
<script src="utils.js"></script>
```

2.3. L'editor de codi

Mentre que Processing incorpora el seu propi editor de codi, p5.js no. Així doncs, necessitarem un editor.

2.3.1. Editor local

Un editor de codi és un editor de text que incorpora funcionalitats específiques que ens fan més fàcil programar, com per exemple el l'acolorit automàtic de diferents parts del codi. Així doncs, si bé podríem editar els arxius del nostre programa amb un editor de text pla, és pràcticament imprescindible fer servir un editor específic per a codi.

Aquestes són algunes opcions gratuïtes i de codi lliure:

- Brackets (http://brackets.io/): Windows, OSX, Linux.
- Atom (https://atom.io/): Windows, OSX, Linux.
- Notepad++ (https://notepad-plus-plus.org/): Windows. Més senzill que els anteriors, però amb tot el que necessiteu.

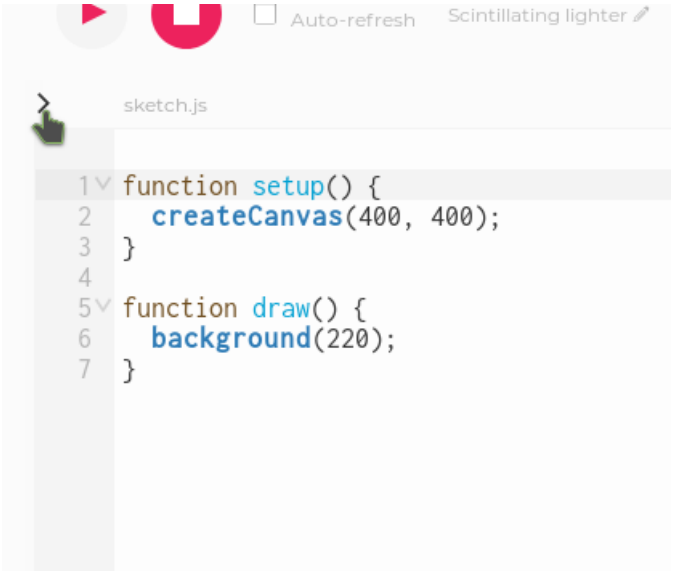
Quan treballem d'aquesta manera, per veure el resultat del nostre programa haurem d'obrir el fitxer 'index.html' amb el navegador web i tornar a carregar la pàgina cada vegada que modifiquem algun arxiu.

2.3.2. L'editor online de p5.js

Una altra opció és fer servir l'editor online de p5.js (https://editor.p5js.org/). Ens proporciona una experiència més similar a l'editor de Processing i és molt còmode si volem començar a fer proves sense haver de preparar un projecte nou. Si ens hi registrem, podem guardar els nostres *sketchs* i accedir-hi des de qualsevol lloc. D'altra banda, si hem d'afegir recursos al nostre projecte (imatges, tipografies, etc.), no és tan còmode com si tenim els arxius al nostre propi ordinador, ja que els hem de pujar cada vegada que els modifiquem i la interfície no és especialment fluida.

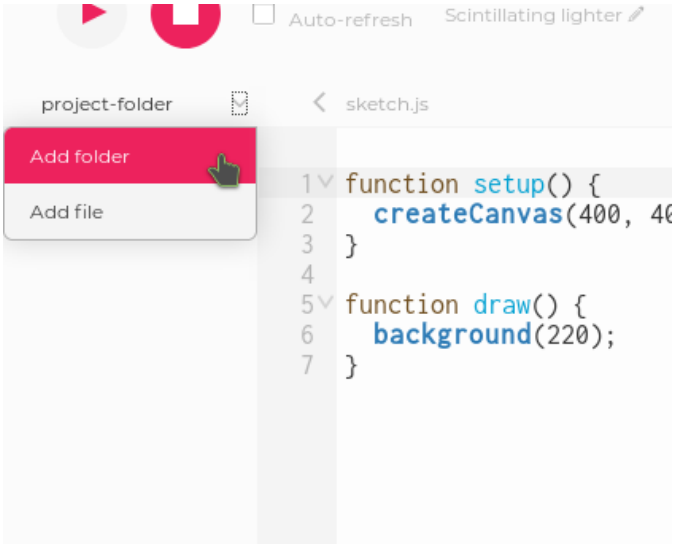
Si, tanmateix, volem pujar arxius a l'editor en líni de p5.js, primer de tot haurem d'estar registrats. Llavors haurem de fer clic a la fletxa que trobem sobre l'espai per al text per desplegar l'inspector de fitxers.

Figura 6. Fletxa per a desplegar l'inspector de fitxers a l'editor online de p5.js



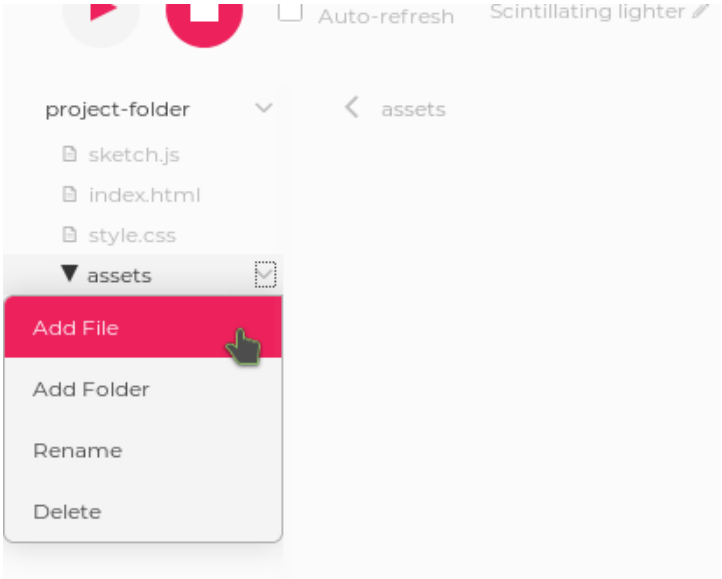
A continuació, si cliquem a la fletxa al costat de 'project-folder' se'ns obre un menú que ens permet crear una carpeta o afegir fitxers. Creem una carpeta 'assets'.

Figura 7. Menú per afegir carpetes i fitxers al projecte de l'editor online de p5.js



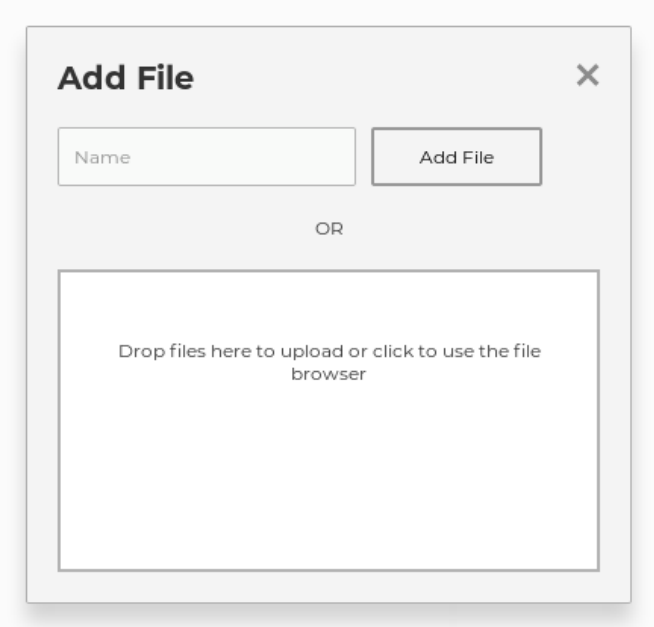
Ara seleccionem la carpeta que acabem de crear i fem clic a la fletxa que hi ha a la seva dreta. Seleccionem 'Add File'.

Figura 8. Afegim arxius a la carpeta que hem creat a l'editor online de p5.js



En el diàleg que ens apareix, fem clic al requadre inferior per seleccionar els arxius que volem pujar o els arrosseguem i els deixem anar a sobre.

Figura 9. Diàleg per afegir fitxers a l'editor online de p5.js



Un cop s'han acabat de pujar els fitxers, podem tancar el diàleg i tornar a seleccionar 'sketch.js' per seguir-lo editant.

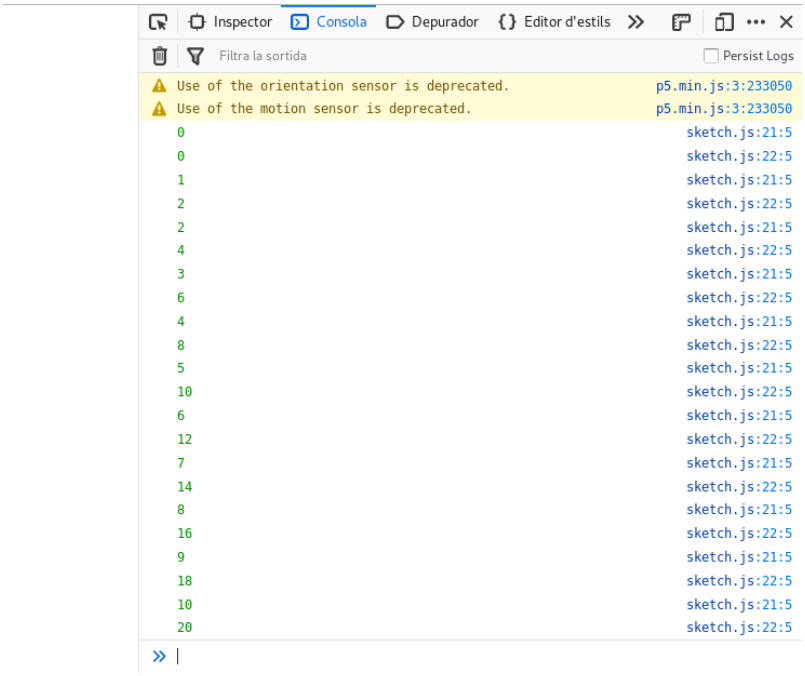
3. La consola de JavaScript

Mentre estem desenvolupant el nostre codi, ens pot resultar útil mostrar el valor de determinades variables. Per això podem fer servir la consola de JavaScript.

La consola de JavaScript és una utilitat dels navegadors web. Segons el nostre navegador la podem obrir de lamanera següent:

- **Firefox:** A 'Eines' > 'Desenvolupador web' > 'Consola web'.
- **Chrome:** Al menú > 'Més eines' > 'Eines per a desenvolupadors', i seleccionem la pestanya 'Console'.
- **Edge:** Premem la tecla F12 i seleccionem la pestanya 'Console'.
- **Safari:** Primer hem de tenir activades les opcions de desenvolupador. Per fer-ho anem a 'Safari' > 'Preferències', i a la pestanya 'Avançat' marquem l'opció "Mostra les opcions de desenvolupador". Un cop fet això podem anar al menú 'Desenvolupador' > 'Mostra la consola de JavaScript'.

Figura 10. Exemple de consola de JavaScript



Per mostrar variables a la consola, p5.js ens facilita la funció ‘print’. Nosaltres recomanem fer servir ‘console.log’ en lloc de ‘print’, ja que és equivalent i és la manera pròpia de JavaScript de fer-ho.

```
function setup() {
	var text = "Hello World!";
	print( text ); // Mostra "Hello World!" a la consola
	console.log( text ); // També mostra "Hello World!" a la consola
}
```

A més, la funció ‘print’ no funciona igual fora de les funcions pròpies de p5.js (‘preload’, ‘setup’, ‘draw’, etc.), ja que serveix per obrir el diàleg de la impressora. ‘console.log’, en canvi, es pot fer servir a tot arreu:

```
var text = "Hello World!";
print( text ); // Obre el diàleg d'impressió
console.log( text ); // Mostra "Hello World!" a la consola
function setup() {
	print( text ); // Aquí sí, mostra "Hello World!" a la consola
}
```

Veureu que a la consola a vegades també hi apareixen altres missatges, com ara errors i avisos del navegador. Alguns avisos, com els que es mostren a la Figura 10, pot ser que us apareguin sempre. No cal que en feu cas. Però si el vostre codi no funciona, pot ser que a la consola s’hi mostri algun error que us ajudi a saber què ha passat.

Si feu servir l’editor en línia de p5.js, veureu que ja duu una consola incorporada. Per tant, no és necessari que obriu la consola del navegador. De totes maneres, a vegades la informació que hi apareix és més completa i ens és més còmode fer-la servir.

4. *let* i *var*

Quan consulteu exemples de codi de p5.js (i JavaScript en general), de vegades trobareu les variables declarades amb la paraula *let* i altres amb la paraula *var*. Generalment aquestes dues paraules són intercanviables, tot i que *let* forma part d'una sintaxi més recent i pot ser que alguns navegadors més antics no la suportin.

La principal diferència entre les dues és en l'àmbit: la zona en la qual la variable existeix un cop declarada. L'àmbit de *var* són les funcions, és a dir, que no podem accedir a aquella variable des de fora de la funció on l'hem declarada. L'àmbit de *let* són els blocs. Un bloc és qualsevol fragment de codi encapsulat entre { }. La declaració d'una funció també constitueix un bloc, però no és l'únic cas.

Per exemple, el codi següent seria vàlid:

```
...
if ( foo > 0) {
     var text = "Text 1";
}
else {
     var text = "Text 2";
}
console.log( text );
```

Mentre que, en aquest altre cas, la consola mostraria 'undefined' o ens donaria un error:

```
...
if ( foo > 0) {
     let text = "Text 1";
}
else {
     let text = "Text 2";
}
console.log( text );// text no existeix
```

Així doncs, l'àmbit de *let* és més restrictiu que el de *var*.

Declarar variables dins de condicionals si les volem fer servir fora del condicional no és recomanable, sinó que és més aconsellable declarar-les abans i assignar-los el valor dins el condicional. D'aquesta manera, és més evident que volem fer servir aquella variable fora del condicional i evitem, per exemple, el descuit d'eliminar la declaració de la variable pensant-nos que no la fem servir més endavant. Pot ser que en projectes senzills això no passi, però quan treballem en projectes més complexos on potser fem diverses coses dins el condicional, aquests descuits són molt corrents.

Per tant, reescriuríem l'exemple anterior de la manera següent:

```
...
let text;
if ( foo > 0) {
     text = "Text 1";
}
else {
     text = "Text 2";
}
console.log( text );
```

Si fem servir *let*, per tant, ens obliguem a seguir bones pràctiques en escriure codi, i així serà més fàcil de llegir i de mantenir.

Hi ha una tercera manera de declarar variables que és mitjançant la paraula *const*. És equivalent a usar *let*, però amb la diferència que no se li pot reassignar un nou valor; la paraula *const* ve de 'constant'. No l'hem usada per no fer el codi més complex, però si us interessa podeu buscar-ne informació, ja que l'ús millora la legibilitat del codi.

5. Objectes

El codi del logotip de *Mosaic* fa servir un ús extensiu d'objectes. Els objectes ens permeten agrupar diferents valors en una única variable. A diferència dels 'arrays', aquests valors s'identifiquen amb noms en lloc de números. És com tenir variables dins d'una variable.

Cadascun dels valors d'un objecte s'anomena *propietat*. JavaScript ens permet crear objectes genèrics als quals podem assignar les propietats que vulguem.

Així doncs, podem crear l'objecte següent:

```
let punt = {
        x: 1,
        y: 3,
        nom: "A",
};
```

Com veieu, podem crear un objecte definint les propietats entre claus ({ }). Per a cada propietat escrivim el nom, dos punts i el valor. Com les variables de JavaScript, les propietats dels objectes poden contenir qualsevol valor de qualsevol tipus.

Llavors podem accedir a les propietats, modificar-les i fins i tot afegir-ne de la manera següent:

```
...
console.log( punt.nom ); // Mostra "A" a la consola
punt.y = 2; // Modifiquem la propietat 'y'
punt.z = -1; // Afegim una propietat 'z'
```

També podem crear objectes buits als quals podem afegir propietats posteriorment:

```
let obj = {};
obj.nom = "A";
```

Una altra sintaxi per accedir a les propietats dels objectes és la següent:

```
let persona = {
        nom: "Mireia",
        edat: 32,
};
console.log( persona[ "nom" ] );
persona[ "edat" ] += 1;
```

Fixeu-vos que en aquest cas hem d'escriure el nom de la propietat entre cometes. També podem guardar, però, el nom de la propietat en una variable:

```
let persona = {
        nom: "Mireia",
        edat: 32,
};
let propietat = "nom";
console.log( persona[ propietat ] ); // Mostra "Mireia" a la consola
```

En aquest exemple tan senzill aquesta sintaxi resulta massa enrevessada, però ens és molt útil si volem accedir a les propietats d'un o diversos objectes de manera programàtica.

6. *class*

En el codi del logotip de *Mosaic* topareu amb una altra sintaxi amb la qual potser no esteu familiaritzats: la declaració *class*.

Si heu fet servir Processing, pot ser que us hàgiu trobat amb el seu equivalent en el llenguatge Java. El funcionament és similar.

La declaració *class* ens permet crear un patró per definir nous objectes. Si heu llegit l'apartat "Objects" del llibre *Make: Getting Started with p5.js*, deveu haver vist que fan servir funcions per crear objectes que responen a un patró. Amb *class* aconseguim el mateix resultat i tècnicament és més adequat. A més, és més fàcil de llegir.

Veiem un exemple:

```
class Counter {
	constructor( incr ) {
		this.val = 0;
		this.incr = incr;
	}
	tick() {
		 this.val += this.incr;
	}
}
let count1 = new Counter(1);
let count2 = new Counter(2);
function setup() {
	frameRate(2);
}
function draw() {
	console.log( count1.val );
	console.log( count2.val );
	count1.tick(); // Incrementem la propietat val de count1
	count2.tick(); // Incrementem la propietat val de count2
}
```

Si executeu aquest codi la consola anirà mostrant els valors següents: 0, 0, 1, 2, 2, 4, 3, 6, 4, 8…

Tenim dos comptadors: un que avança d'un en un i l'altre que avança de dos en dos. Però podríem crear tots els que volguéssim amb els increments que volguéssim.

Com podeu veure, la declaració *class* consta d'un nom de classe (en aquest cas 'Counter') i d'una sèrie de mètodes. Fixeu-vos que escrivim la primera lletra dels noms de les classes en majúscula, per tal de diferenciar-les dels noms de les variables.

Anomenem *mètodes* les funcions que formen part d'un objecte o una classe. Però quan definim una classe, no hem d'escriure la paraula *function* davant dels mètodes.

El primer mètode de tots, 'constructor', és un mètode especial i és el principal de les classes, ja que és el que s'executa quan creem una nova instància de la classe: quan creem un nou objecte. Per fer-ho fem servir l'operador *new*:

```
...
let count1 = new Counter(1);
let count2 = new Counter(2);
...
```

Com podeu veure, passem a la classe els arguments que hem definit a 'constructor', ja que el que passa en realitat quan fem servir l'operador *new* és que s'executa el mètode 'constructor' i es retorna l'objecte creat. Fixeu-vos que a dins de 'constructor' fem servir la variable 'this'.

```
...
        constructor( incr ) {
                this.val = 0;
                this.incr = incr;
        }
...
```

'this' fa referència a l'objecte que estem creant. De la mateixa manera, en qualsevol dels altres mètodes de la classe podem fer referència a la variable 'this' per accedir a les propietats i als altres mètodes de la instància.

```
...
        tick() {
                 this.val += this.incr;
        }
...
```

Així, un cop creem una instància de la classe, podem accedir als seus mètodes, que només afectaran les propietats d'aquella instància. Com veieu, també podem accedir directament a les seves propietats:

```
...
        console.log( count1.val );
        console.log( count2.val );
        count1.tick(); // Incrementem la propietat val de count1
        count2.tick(); // Incrementem la propietat val de count2
...
```

La sintaxi de *class* és relativament recent i pot ser que no funcioni en alguns navegadors més antics. Hi ha una altra manera d'aconseguir el mateix resultat, els 'prototips', que són la manera tradicional de fer-ho en JavaScript. El funcionament pot resultar més complicat d'entendre, tot i que en realitat *class* fa servir de manera interna els prototips. Els prototips són una faceta molt flexible de JavaScript de la qual podeu buscar més informació si us interessa.

7. Integració amb Wordpress

Per integrar el nostre projecte en un Wordpress farem servir l'extensió 'p5.js Embed'. Llegiu la guia d'ús corresponent per saber com fer-la servir.

'p5.js Embed' ens proporciona una variable 'wp_data', on trobem informació relativa al lloc web i a la pàgina que s'està consultant. Si treballem en el nostre *sketch* de manera aïllada no disposarem d'aquesta variable. Si volem que el nostre codi funcioni tant dins com fora de Wordpress, podem comprovar si la variable 'wp_data' existeix i, en cas que no sigui així, fer servir una sèrie de valors per defecte. Per exemple:

```
let post_count;
```

```
if ( typeof( wp_data ) !== "undefined" ) {
       post_count = wp_data.post_count;
}
else {
       post_count = 10;
}
```

'typeof' és una funció de JavaScript que retorna el tipus d'una variable. Quan la variable no ha estat definida retorna com a "undefined".