Unidad 2

Formas: Coordinadas y Primitivas

Elementos que se introducen en esta Unidad:
`size(), point(), line(), triangle(), quad(), rect(), ellipse(), bezier(), background(), fill(), stroke(), noFill(), noStroke(), strokeWeight(), strokeCap(), strokeJoin(), smooth(), noSmooth(), ellipseMode(), rectMode()`

Dibujar una forma con un código puede ser difícil porque todos sus aspectos de su ubicación deben ser especificados con un número. Cuando uno está acostumbrado a dibujar con un lápiz o formas moviéndose en una pantalla con el ratón, puede tomar mucho tiempo empezar a pensar en la relación de la red con una pantalla de coordenadas estrictas. La diferencia fundamental entre ver una composición sobre papel o en su mente y su traducción en notación de código es muy amplia, pero muy sencilla de entender.

-Coordenadas

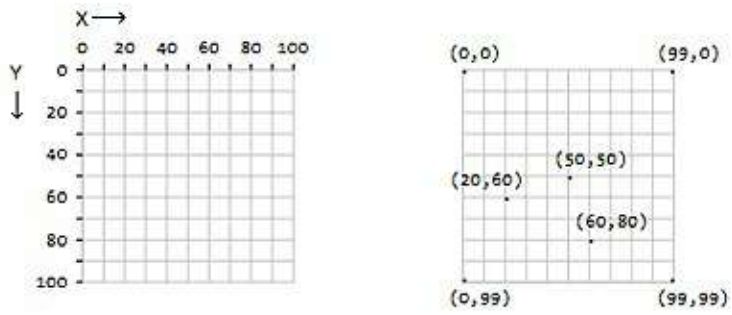
Antes de hacer un dibujo, es importante que pensemos acerca del tamaño y las características de la superficie sobre la que vamos a dibujar. Si vamos a hacer un dibujo en papel, podemos elegir multitud de utensilios y tipos de papel. Para un esbozo rápido, papel de periódico y carboncillo es lo más apropiado. Para un dibujo más refinado, puede ser preferible papel suave hecho a mano y lápices. Contrariamente, cuando estás dibujando en un ordenador, las opciones principales disponibles son el tamaño de la ventana y el color del fondo.
La pantalla de un ordenador es una **rejilla** de pequeños elementos luminosos llamados píxeles. Las pantallas vienen en muchos tamaños y resoluciones. Existen tres tipos diferentes de pantallas para nuestro estudio, y todas ellas tienen un número diferente de píxeles. Los portátiles tienen 1.764.000 píxeles (1680 de ancho por 1050 de alto), las pantallas planas tienen 1.310.720 píxeles (1280 de ancho por 1024 de alto) y los viejos monitores tienen 786.432 píxeles (1024 de ancho por 768 de alto). Millones de píxeles pueden sonar como una cantidad muy vasta, pero producen una pobre calidad visual comparado a un medio físico, como el papel. Las pantallas modernas tienen una resolución de aproximadamente cien puntos por pulgada, mientras que las impresoras modernas proveen más de mil puntos por pulgada. Por otra parte, las imágenes en papel son fijas, mientras que las pantallas tienen la ventaja de ser capaces de cambiar su imagen muchas veces por segundo. Los programas en Processing pueden controlar **todos** o un **subconjunto** de píxeles de la pantalla. Cuando pulsa el botón Run, una ventana de representación se abre y te permite leer y escribir dentro de los píxeles. Es posible crear imágenes más grandes que la pantalla, pero en la mayoría de los casos, haremos una ventana de representación igual o menor a la pantalla. El tamaño de la ventana de representación está controlada por la función `size()`:

```
size(ancho, alto)
```

La función `size()` tiene dos parámetros: el primero establece el ancho de la ventana y el segundo su alto.

```
//Dibuja la ventana de representación de 320 de ancho y 240 de alto (píxeles).
size(320,240);
```

Una posición de la pantalla está comprendida por un eje de coordenadas x y un eje de coordenadas y. El eje de coordenadas x es la distancia horizontal desde el origen y el eje de coordenadas y es la distancia vertical. En Processing, el origen es la **esquina superior izquierda** de la ventana de representación y coordina los valores hacia abajo y hacia la derecha. La imagen de la izquierda muestra el sistema de coordenadas, y la imagen de la derecha muestra varias posiciones en la rejilla:



Una posición se escribe **con el valor del eje x seguido del valor del eje y**, separados por una coma. La notación para el origen es `(0,0)`, la coordenada `(50,50)` tiene 50 de coordenada x y 50 de coordenada y, y la coordenada `(20,60)` tiene 20 de coordenada x y 60 de coordenada y. Si el tamaño de la ventana de representación es de 100 píxeles de ancho y 100 píxeles de alto, el píxel de la esquina superior izquierda es `(0,0)`, el píxel de la esquina superior derecha es `(99,0)`, el píxel de la esquina inferior izquierda es `(0,99)`, y el píxel de la esquina inferior derecha es `(99,99)`. Esto se ve más claro si usamos la función `point()`.

-Figuras primitivas

Un punto es el elemento visual más simple y se dibuja con la función `point()`:

```
point(x,y)
```

Esta función tiene dos parámetros: el primero es la coordenada x y el segundo es la coordenada y. A menos que se especifique otra cosa, un punto es del tamaño de un sólo píxel.

```
point(20, 20);
point(30, 30);
point(40, 40);
point(50, 50);
point(60, 60);
```

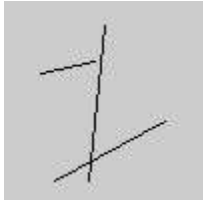


```
point(-500, 100); //Los parámetros negativos no provocan
point(400, -600); //error, pero no se verán en la ventana de
point(140, 2500); //representación.
point(2500, 100);
```

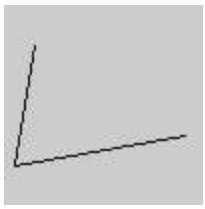
Es posible dibujar cualquier línea mediante una serie de puntos, pero son más simples de dibujar con la función `line()`. Esta función tiene cuatro parámetros, dos por cada extremo:

```
line(x1, y1, x2, y2)
```

Los primeros dos parámetros establecen la posición donde la línea empieza y los dos últimos establecen la posición donde la línea termina.



```
line(25, 90, 80, 60);
line(50, 12, 42, 90);
line(45, 30, 18, 36);
```



```
line(15, 20, 5, 80);
line(90, 65, 5, 80);
```

La función `triangle()` dibuja un triángulo. Tiene seis parámetros, dos por cada punto:

```
triangle(x1, y1, x2, y2, x3, y3)
```

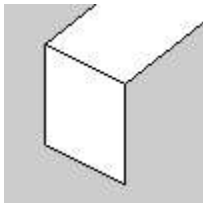
El primer par define el primer punto, el segundo par el segundo punto y el último par el tercer punto. Cualquier triángulo puede ser dibujado conectando tres líneas, pero la función `triangle()` hace posible dibujar una figura con relleno. Triángulos de todas formas y tamaños pueden ser creados cambiando los valores de los parámetros.

```
triangle(55, 9, 110, 100, 85, 100);
triangle(55, 9, 85, 100, 75, 100);
triangle(-1, 46, 16, 34, -7, 100);
triangle(16, 34, -7, 100, 40, 100);
```

La función `quad()` dibuja un cuadrilátero, un polígono de cuatro lados. Esta función tiene ocho parámetros, dos por cada punto.

```
quad(x1, y1, x2, y2, x3, y3, x4, y4)
```

Variando los valores de los parámetros se puede construir rectángulos, cuadrados, paralelogramos y cuadriláteros irregulares.



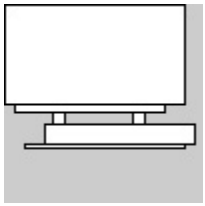
```
quad(20, 20, 20, 70, 60, 90, 60, 40);
quad(20, 20, 70, -20, 110, 0, 60, 40);
```

Dibujar rectángulos y elipses funcionan de una manera diferente a las figuras vistas anteriormente. En lugar de definir cada punto, los cuatro parámetros establecen la posición y las dimensiones de la figura. La función `rect()` dibuja un rectángulo:

```
rect(x, y, ancho, alto)
rect(x, y, ancho, alto, radio)                //A partir de la versión 2.0
rect(x, y, ancho, alto, si, sa, id, ii)       //A partir de la versión 2.0
```

Los dos primeros parámetros establecen la localización de la esquina superior izquierda, el tercero establece el ancho, y el cuarto el alto. Use el mismo valor de `ancho` y `alto` para dibujar un cuadrado. La versión con el parámetro `radio` permite agregar borde redondeados al rectángulo. El parámetro `radio` corresponde al radio de redondeo. La versión con otros cuatro parámetros permite agregar un radio distinto de redondeo a

cada esquina. Siendo `si` (Superior Izquierdo), `sa` (Superior Derecho), `id` (Inferior Derecho) y `ii` (Inferior Izquierdo).

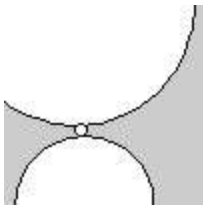


```
rect(0, 0, 90, 50);
rect(5, 50, 75, 4);
rect(24, 54, 6, 6);
rect(64, 54, 6, 6);
rect(20, 60, 75, 10);
rect(10, 70, 80, 2);
```

La función `ellipse()` dibuja una elipse en la ventana de representación:

```
ellipse(x, y, ancho, alto)
```

Los dos primeros parámetros establecen la localización del centro de la elipse, el tercero establece la ancho, y el cuarto la altura. Use el mismo valor de `ancho` y de `alto` para dibujar un círculo.

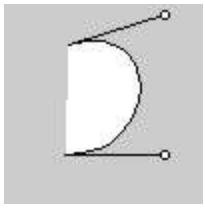


```
ellipse(35, 0, 120, 120);
ellipse(38, 62, 6, 6);
ellipse(40, 100, 70, 70);
```

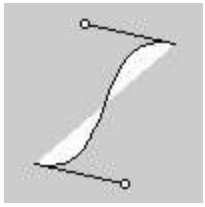
La función `bezier()` puede dibujar líneas que no son rectas. Una curva Bézier está definida por una serie de puntos de control y puntos de anclaje. Una curva es dibujada entre dos puntos de anclaje, y los puntos de control definen su forma:

```
bezier(x1, y1, cx1, cy1, cx2, cy2, x2, y2)
```

Esta función requiere ocho parámetros para definir cuatro puntos. La curva se dibuja entre el primer punto y el cuarto, y los puntos de control están definidos por el segundo y tercer punto. En los programas que se utilizan curvas Bézier, tales como Adobe Illustrator o Inkscape, los puntos de control son representados por pequeños nodos que sobresalen de los bordes de la curva.



```
bezier(32, 20, 80, 5, 80, 75, 30, 75);
//Dibujamos los puntos de control.
line(32, 20, 80, 5);
ellipse(80, 5, 4, 4);
line(80, 75, 30, 75);
ellipse(80, 75, 4, 4);
```

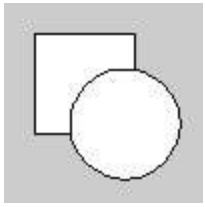


```
bezier(85, 20, 40, 10, 60, 90, 15, 80);
//Dibujamos los puntos de control.
line(85, 20, 40, 10);
ellipse(40, 10, 4, 4);
line(60, 90, 15, 80);
ellipse(60, 90, 4, 4);
```

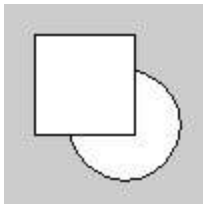
-Orden de dibujo

El orden en que dibujamos las figuras en el código, define qué figuras aparecerán sobre otras en la ventana de representación. Si dibujamos un rectángulo en la primera línea de un programa y una elipse en la segunda línea, el rectángulo aparecerá debajo de la elipse cuando ejecutemos el programa. Revirtiendo el orden, el

rectángulo se coloca arriba.



```
rect(15, 15, 50, 50);         //Abajo
ellipse(60, 60, 55, 55);      //Arriba
```

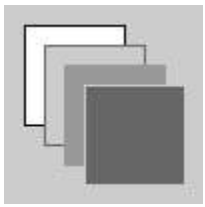


```
ellipse(60, 60, 55, 55);      //Abajo
rect(15, 15, 50, 50);         //Arriba
```

-Valores de grises

Los ejemplos vistos anteriormente han usado el fondo por defecto de color gris claro, líneas negras, y figuras blancas. Para cambiar estos valores, es necesario introducir sintaxis adicional. La función `background()` establece el color de la ventana de representación con un número entre 0 y 255. Este rango puede ser incómodo si no estás familiarizado con programas de dibujo en el ordenador. El valor 255 es blanco y el valor 0 es negro, con un rango de valores de grises en medio. Si no se define un valor para el fondo, se usa el valor por defecto 204 (gris claro).

La función `fill()` define el valor del relleno de las figuras, y la función `stroke()` define el valor del contorno de las figuras dibujadas. Si no se define un valor de relleno, se usa el valor por defecto `255` (blanco). Si no se define un valor de contorno, se usa el valor por defecto `0` (negro).



```
rect(10, 10, 50, 50);
fill(204);
stroke(102);
rect(20, 20, 50, 50);
fill(153);
stroke(153);
rect(30, 30, 50, 50);
fill(102);
stroke(204);
rect(40, 40, 50, 50);
```

Cuando se ha definido un valor de relleno o contorno, se aplica a todas las figuras dibujadas después. Para cambiar el valor de relleno o contorno, usamos la función `fill()` o `stroke()` de nuevo.

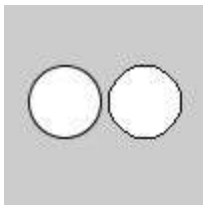
Un parámetro opcional adicional para `fill()` o `stroke()` regula la transparencia. Definiendo el parámetro a `255` hace que la figura sea totalmente opaca, y a `0` totalmente transparente:

```
background(0);
fill(255, 220);
rect(15, 15, 50, 50);
rect(35, 35, 50, 50);
```

El relleno y el contorno de una figura se puede eliminar. La función `noFill()` detiene a Processing de rellenar figuras, y la función `noStroke()` detiene la creación de líneas y contornos de las figuras. Si usamos `noFill()` y `noStroke()` no dibujaremos nada en la pantalla.

-Atributos de dibujo

Además de cambiar los valores de relleno y contorno de las figuras, también es posible cambiar atributos de la geometría. Las funciones `smooth()` y `noSmooth()` activan y desactivan el suavizado (conocido como filtro *antialiasing*). Cuando usamos una de estas funciones, afectará a todas las funciones dibujadas después. Si usamos primero `smooth()`, usar `noSmooth()` cancelará el ajuste, y viceversa.



```
smooth();
ellipse(30, 48, 36, 36);
noSmooth();
ellipse(70, 48, 36, 36);
```

Los atributos de la línea, están controlados por las funciones `strokeWeight()`, `strokeCap()` y `strokeJoin()`. La función `strokeWeight()` tiene un parámetro numérico que define el grosor de todas las líneas dibujadas después de usar esta función. La función `strokeCap()` requiere un parámetro que puede ser `ROUND`, `SQUARE` o `PROJECT`. `ROUND` redondea los puntos finales, y `SQUARE` los cuadra. `PROJECT` es una mezcla de ambos: redondea las esquinas cuadradas suavemente. Esta función se usa en líneas. La función `strokeJoin()` tiene un parámetro que puede ser `BEVEL`, `MITTER` o `ROUND`. Estos parámetros determinan la forma del contorno de la figura. `BEVEL` corta en diagonal las esquinas cuadradas, `MITTER` cuadra las esquinas (es el valor por defecto) y `ROUND` redondea las esquinas.

```
smooth();
strokeWeight(12);
strokeCap(ROUND);
line(20, 30, 80, 30);   //Línea superior
strokeCap(SQUARE);
line(20, 50, 80, 50);   //Línea central
strokeCap(PROJECT);
line(20, 70, 80, 70);   //Línea inferior
```

```
smooth();
strokeWeight(12);
strokeJoin(BEVEL);
rect(12, 33, 15, 33);   //Figura izquierda
strokeJoin(MITER);
rect(42, 33, 15, 33);   //Figura central
strokeJoin(ROUND);
rect(72, 33, 15, 33);   //Figura derecha
```

-Modos de dibujo

Por defecto, los parámetros para `ellipse()` definen la coordenada x del centro, la coordenada y del centro, el ancho y el alto. La función `ellipseMode()` cambia la forma en que se usan estos parámetros para dibujar elipses. La función `ellipseMode()` requiere un parámetro que puede ser `CENTER`, `RADIUS`, `CORNER` o `CORNERS`. El modo por defecto es `CENTER`. El modo `RADIUS` también usa el primer y segundo parámetro de `ellipse()` para establecer el centro, pero el tercer parámetro deber ser la mitad del ancho, y el cuarto parámetro debe ser la mitad del alto. El modo `CORNER` hace que `ellipse()` funcione de manera parecida a `rect()`. Causa que el primer y segundo parámetro se trasladen a la esquina superior izquierda del rectángulo que circunscribe la elipse y usa el tercer y cuarto parámetro para definir la anchura y la altura. El modo `CORNERS` tiene un efecto similar a `CORNER`, pero causa que el tercer y cuarto parámetro de `ellipse()` se definan en la esquina inferior derecha del rectángulo.

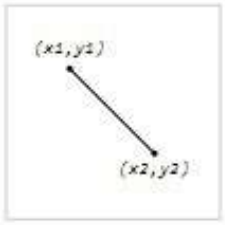
```
smooth();
noStroke();
ellipseMode(RADIUS);
fill(126);
ellipse(33, 33, 60, 60);      //Elipse gris
fill(255);
ellipseMode(CORNER);
ellipse(33, 33, 60, 60);      //Elipse blanca
fill(0);
ellipseMode(CORNERS);
ellipse(33, 33, 60, 60);      //Elipse negra
```

Con un estilo similar, la función `rectMode()` afecta a cómo se dibujan los rectángulos. Requiere un parámetro que puede ser `CORNER`, `CORNERS` o `CENTER`. El modo por defecto es `CORNER`, y `CORNERS` causa que el tercer y cuarto parámetro de `rect()` se definan en la esquina contraria a la primera. El modo `CENTER` causa que el primer y segundo parámetro de `rect()` definan el centro del rectángulo y usa el tercer y cuarto parámetro para el ancho y el alto.

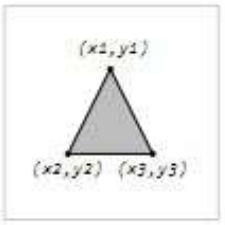
```
noStroke();
rectMode(CORNER);
fill(126);
rect(40, 40, 60, 60);         //Rectángulo gris
rectMode(CENTER);
fill(255);
rect(40, 40, 60, 60);         //Rectángulo blanco
rectMode(CORNERS);
fill(0);
rect(40, 40, 60, 60);         //Rectángulo negro
```

Apéndice de Primitivas

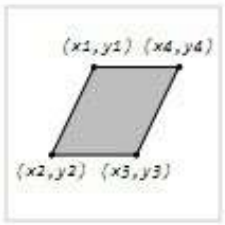

```
point(x, y)
```



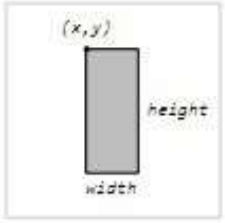

```
line(x1, y1, x2, y2)
```



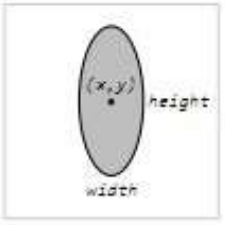

```
triangle(x1, y1, x2, y2, x3, y3)
```



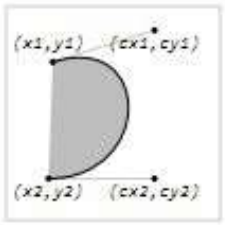

```
quad(x1, y1, x2, y2, x3, y3, x4, y4)
```




```
rect(x, y, width, height)
```




```
ellipse(x, y, width, height)
```




```
bezier(x1, y1, cx1, cy1, cx2, cy2, x2, y2)
```