TEMA 4: PROGRAMACIÓN

1. Introducción

Un **programa** es una secuencia de instrucciones escritas mediante un lenguaje de programación, que permiten desarrollar y resolver un problema para el que han sido creados.

Un programa se compone de:

- **Algoritmo:** conjunto de instrucciones que describen el proceso que se debe seguir, para dar solución a un problema específico. Es una secuencia ordenada de pasos o instrucciones.

Programar es escribir y desarrollar un programa utilizando un **lenguaje de programación.** El lenguaje de programación es el conjunto de símbolos y palabras que definen ese "idioma" o lenguaje.

Existen muchos lenguajes de programación, cada uno tiene su propio vocabulario (órdenes) y sintaxis (forma de colocar las órdenes)

Ejemplos de Lenguajes de Programación	
Lenguaje	**Aplicación**
HTML	Creación de páginas web
Java	Creación de juegos y aplicaciones
JavaScript	Creación de programas interactivos
C	Creación de sistemas operativos
Processing	Creación de programas multimedia
Arduino, Winlogo, Robolab	Robótica

Los programas se escriben en **código fuente** (mediante un lenguaje de programación). Este código debe traducirse a código binario, llamado **código máquina**, que es el único lenguaje que entiende el ordenador. Este proceso de traducción se llama **compilación**.

Para poder escribir nuestros programas necesitamos un **IDE** (Entorno de Desarrollo Integrado), que nos permita teclear las instrucciones, compilarlas y ejecutar el programa.

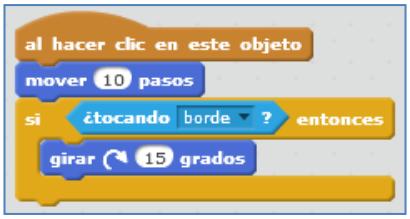
2. Tipos de lenguajes de programación

- **Lenguajes textuales:** las ordenes de programación se insertan en forma de texto. Las ordenes se escriben en inglés, y muchas de ellas coinciden en los diferentes lenguajes (repeat, delay, if, while,....)

  Este tipo de lenguajes son los más potentes y son los utilizados por los profesionales de la programación. Ejs: Phyton, JavaScript, C, Processing

```
int factorial(int x)
{
    if (x == 0)
        return 1;
    else
        return x * factorial(x - 1);
}
```

- **Lenguajes gráficos:** las ordenes o instrucciones se dan mediante bloques gráficos. Ej: Scratch



3. Fases del proceso de programación:

1. **Definición** y análisis del problema
2. **Diseño** del algoritmo
3. **Codificación** del programa: obtenemos el código fuente
4. **Compilación:** obtenemos el código máquina o código objeto
5. **Depuración** de errores y verificación del programa
6. **Mantenimiento**

4. Algoritmo: codigo, pseudocódigo y diagráma de flujo

- **Algoritmo:** conjunto ordenado de instrucciones que describen el proceso que se debe seguir, para desarrollar un programa informático.

  Los algoritmos deben tener las siguientes partes:

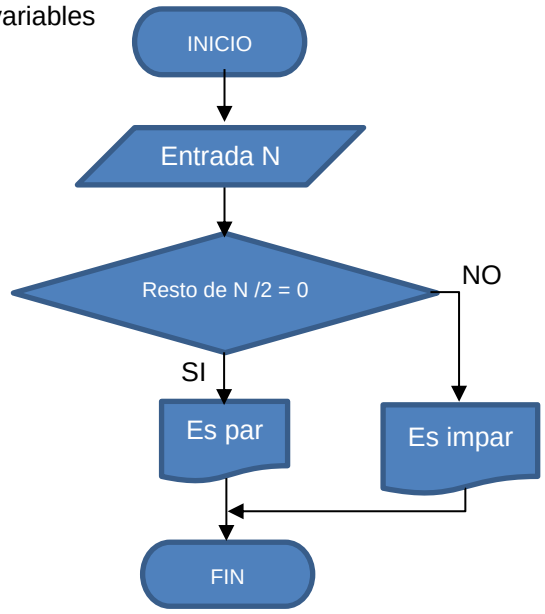
  - **Entrada** de datos
  - **Proceso**: pasos necesarios para ejecutar el algoritmo
  - **Salida** de resultados

- El programa creado mediante lenguaje de programación (serie de instrucciones) se llama **código**.
- Cuando escribimos el algoritmo en nuestra lengua o lenguaje habitual, el programa creado se llama **pseudocódigo.**

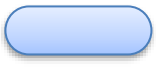
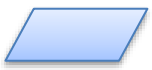
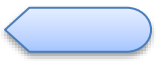
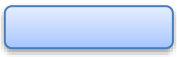
  Cada uno puede utilizar sus propios criterios, pero se recomiendan unas normas generales
- Las instrucciones del programa también se pueden representar gráficamente mediante esquemas o dibujos, llamados **diagramas de flujo.**

  Cada uno puede utilizar sus propios criterios, pero se suelen seguir unas normas generales (utilización de símbolos)

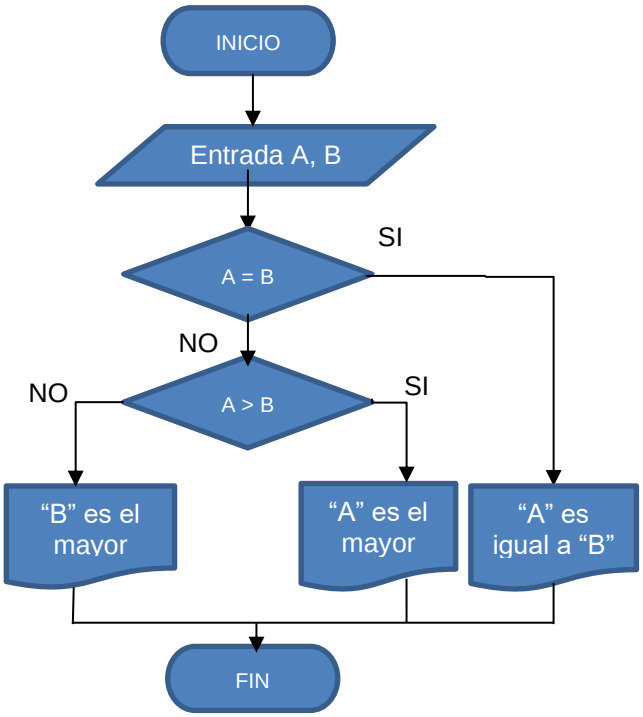
Desarrollar un algoritmo que permita leer un número cualquiera N y escriba si dicho número es par o impar

PSEUDOCÓDIGO	DIAGRAMA DE FLUJO
1. **Declaración de variable**: N **2. Inicio** ▪ Introducir un número N ▪ Condición: Si el resto de dividir N/2 = 0 ▪ **SI**: entonces escribir "es par" ▪ **Si NO**: escribir "es impar" **3. Fin**	En los diagramas no se representa la declaración de variables 

- **Simbolos de diagrams de flujo:**

	**Inicio / Final**		**Decisión**
	**Entrada / Salida de datos**		**Impresora / Documento** (resultados impresos)
	**Entrada de datos por teclado**		**Pantalla** (resultados en pantalla)
	**Acción / Proceso** (instrucción a realizar)		

- **Ejemplos**

Hacer un diagrama de flujo que permita leer 2 números diferentes y nos diga cuál es el mayor de los 2 números.	
DIAGRAMA DE FLUJO	**PSEUDOCÓDIGO**
	1. **Declaración de variables**: • A • B **2. Inicio** ▪ Introducir dos datos A, B ▪ Leer los dos valores ▪ Condición: Si A = B ▪ **Si**: entonces escribir "A" es igual a "B" ▪ **Si NO:** ▪ **Si** A > B: escribir "A es el mayor" ▪ **Si No** escribir "B es el mayor" **3. Fin**

Ejercicios:

1. Hacer el diagrama de flujo y pseudocódigo para realizar la media aritmética de 4 números enteros introducidos por teclado y escribir el resultado.
2. Realizar el diagrama de flujo o pseudocódigo para que nos calcule la hipotenusa de un triángulo rectángulo, conocidos su dos catetos.
3. Crear un diagrama de flujo o pseudocódigo en el que se introducen 3 números A, B y C. El diagrama debe decidir cual es el mayor y cual es el menor

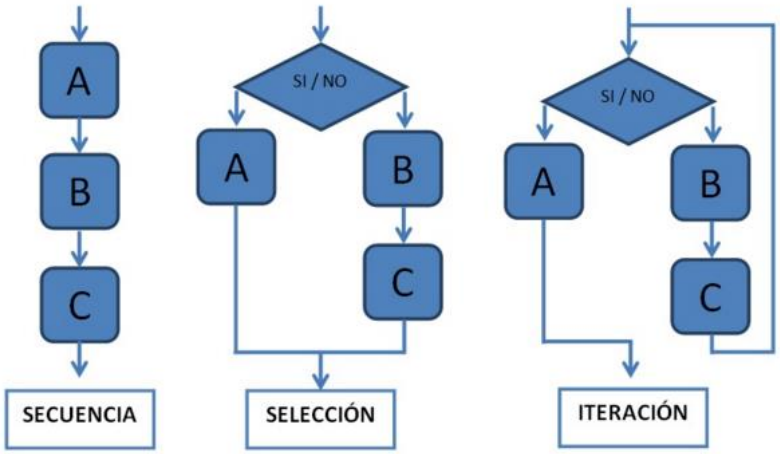
5. Tipos de estructuras de control

Las estructuras de control deciden qué camino hay que seguir en función de una condición. Pueden ser de tres tipos:

- **Estructura secuencial:** consiste en colocar una instrucción tras otra, de manera que se van ejecutando de arriba abajo.
- **Estructura selectiva o condicional (si, si no):** permiten ejecutar un conjunto de instrucciones u otras en función de si se cumple o no una condición.

  Se utilizan los comandos: **if y if…else**
- **Estructura iterativa o de repetición (mientras, repetir, para):** permite repetir una instrucción o grupo de ellas un nº fijo de veces o mientras (o hasta que) una condición sea cierta.

  Se utilizan los comandos **while, do….while, for**



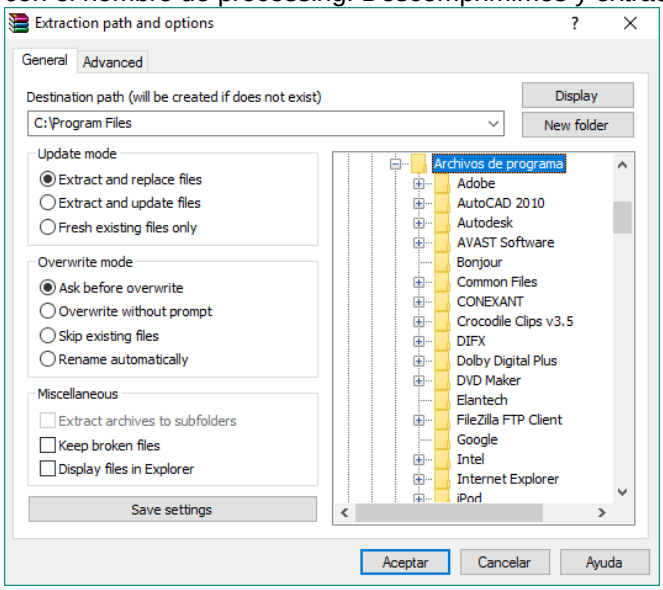
PROGRAMACIÓN CON PROCESSING

Processing es un lenguaje de programación de código abierto. Está orientado para programar proyectos multimedia y para aprender a programar. Está basado en el lenguaje de programación **Java** y se parece mucho a **Wiring,** que es el que utiliza **Arduino.**

6. Instalación de Processing

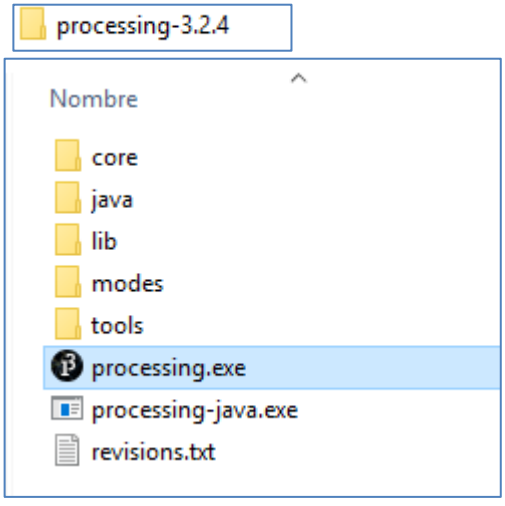
Debemos descargar el programa de la web oficial**:** https://processing.org/

Al abrir el archivo que hemos descargado vemos que es un archivo comprimido. Tenemos una carpeta con el nombre de processing. Descomprimimos y extraemos la carpeta donde queremos que se ejecute.

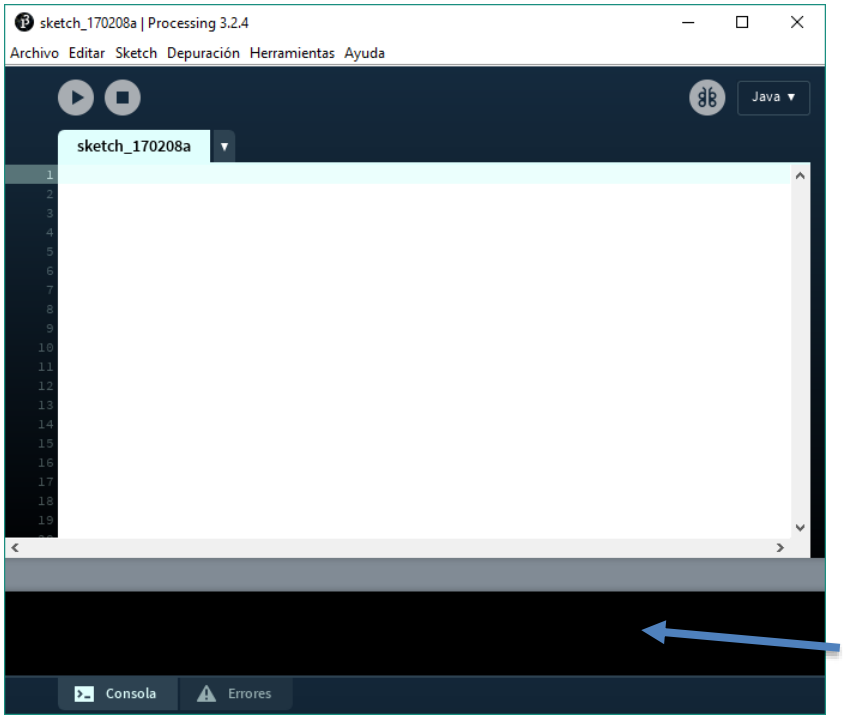


Y obtenemos una carpeta en la que habrá un ejecutable.

Este ejecutable podemos llevarlo al escritorio o anclarlo al menú inicio



7. Entorno Processing



Ejecutar: Ejecuta en la ventana el resultado de la instrucción escrita, es decir, ejecuta el código.
Este proceso realiza la compilación (pasar a código binario).
Al ejecutar cualquier programa, se despliega una ventana. En esta ventana se puede ver el resultado gráfico de la ejecución del programa

Detener: detiene la ejecución del programa

Depurar: revisa el programa y muestra los errores

Consola: muestra los errores del programa

Editar / Autoformato: Da un formato al código que facilita su compresión, lo ordena y lo alinea verticalmente

Cuando creamos un proyecto nuevo en Processing, creamos un **Sketch**, y el **Sketch folder** es la carpeta donde éste se guarda. En principio, el Sketch consta tan solo de la misma carpeta y de un **archivo .pde** que contiene el código. Luego se pueden añadir carpetas para archivos de imagen, video, etc.

RECOMENDACIÓN: Debemos ir guardando diferentes versiones mientras trabajamos en nuestros proyectos con el comando **Guardar como**.

Sketch / Tweak: Esta función permite modificar el código y poder ¡ver los resultados en tiempo real!

Autocompletado: Es complicado aprenderse todas las funciones de un lenguaje de programación, sobretodo porque basta con equivocarse en una letra para que no funcione, esta función te permite escribir parte de la función y terminar de escribirla automáticamente.
Se puede activar en **preferencias**

Sketch / Importar biblioteca: permite añadir las librerías más comunes, y podemos buscar y descargar las librerías hechas por terceros.

8. Sintaxis de Processing

- Las ordenes o **instrucciones** se separan mediante **;**
- Los **valores** de cada instrucción se colocan entre paréntesis **( )**

  Ejemplo: point (100,100);
- **Comentarios** Son usados para explicar el programa, se colocan entre las líneas de código .Se deben iniciar el comentario con **//**

  Ejemplo: //int v - velocidad de desplazamiento.

  Para hacer comentarios que requieren más de una línea, se debe iniciar el comentario con **/*** y finalizarlo con ***/**

  Ejemplo: /* Esta instrucción dibuja 20 líneas y modifica el color según con la posición en el eje y */

  Los comentarios no se ejecutan

- Processing distingue entre mayúsculas y minúsculas, no las confundas.
- Processing no tiene en cuenta los espacios en blanco, da igual ponerlos o no ponerlos.

9. Dibujar con Processing

Cuadrícula

Processing y java, trabajan píxel a píxel. Es decir, para dibujar con Processing hay que tener en cuenta que trabajamos sobre una cuadrícula de píxeles, que puede ser de 100x100 (tamaño por defecto), o el que nosotros definamos.

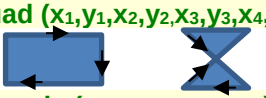
Para dibujar, por ejemplo, una línea, tenemos que especificar en el código de qué punto a qué otro de la cuadrícula tiene que ir. Para una esfera, el punto central y el radio, etc.

Cada píxel, tiene su lugar en la cuadrícula. Esta posición se expresa mediante **coordenadas X, Y**, con el punto 0,0 en la **esquina superior izquierda** de la ventana.

Instrucción	Descripción
size (x,y)	Tamaño de la ventana o pantalla de dibujo

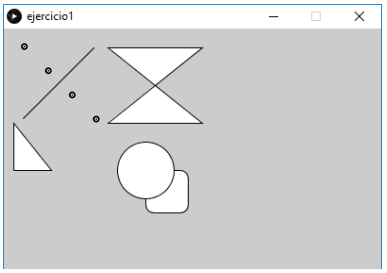
Si después de programar hay que cambiar el tamaño del dibujo, debemos usar **surface.setSize (x, y)**

Funciones de dibujo

Instrucción	Descripción
point (x,y)	**punto** (dibuja un punto con 1px de diámetro)
line (x_1,y_1,x_2,y_2)	**línea** (definida por las coordenadas del punto origen y fin)
rect (x,y,ancho,alto,radio) 	**rectángulo** (definido por las coordenadas x,y de la esquina superior izquierda; ancho y alto; radio es opcional y redondea las esquinas) Para dibujar **rectángulos centrados** podemos cambiar el modo de dibujo **rectMode(CENTER);** **rect(100,100,50,30);** 
quad (x_1,y_1,x_2,y_2,x_3,y_3,x_4,y_4) 	**cuadrilátero** (cada par de coordenadas representa en orden de dibujo, uno de los cuatro vértices). Permite hacer cuadrados, rombos,…
triangle (x_1,y_1,x_2,y_2,x_3,y_3)	**triángulo** (cada par de coordenadas representa en orden de dibujo, uno de los tres vértices).
ellipse (x,y,ancho,alto)	**elipse** (definido por las coordenadas x,y del centro; ancho y alto). Permite dibujar circunferencias

Ejercicio 1:

- Define el tamaño de la ventana de 500x500 px
- Dibuja cuatro puntos desfasados 25 px (en horizontal y vertical)
- Dibuja una linea inclinada simétrica a los 4 puntos dibujados
- Dibuja un triángulo rectángulo de lado 50 px
- Dibuja un rombo de ancho 100px y alto 80px
- Dibuja un cuadrado de lado 45 px, con bordes redondeados y centrado en la ventana
- Dibuja una circunferencia de radio 60px
- Manda el archivo por mail



El mode(CENTER) también puede usarse para las imágenes y las formas

Color

Processing puede trabajar con el modo de color **RGB** utilizando cuatro variables (rojo, verde, azul y alfa o transparencia), definidas cada una por un byte con un rango de valores que va de 0 a 255

color(255,0,0) = rojo **color(0,255,0,127)=** verde con 50% transparencia

Y con el modo de color **hexadecimal : #FF5733, transparencia**

En ambos casos si no se especifica el parámetro alfa, se entiende que su valor es máximo, 255, y que por lo tanto no hay transparencia

Si un color tiene sus tres valores R, G y B iguales entre sí, se puede expresar con un solo valor: **color(130);**

Valor	Colores
color (0)	Negro
color (255)	Blanco
color (255,0,0)	Rojo
color (0,255,0)	Verde
color (0,0,255)	Azul
alfa 255	Sin transparencia
alfa 0	Total transparencia

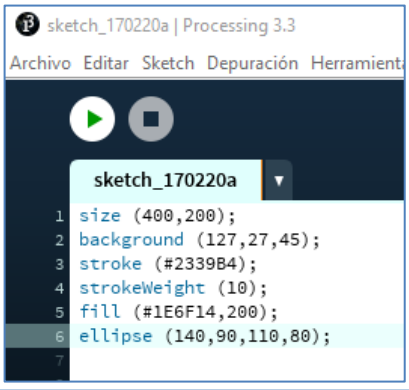
Cada figura en processing tiene dos colores: **stroke, fill**

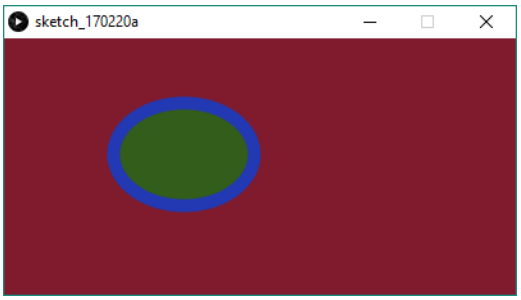
Instrucción	Descripción
stroke (r,g,b,α)	Color RGB del borde α = 0 (transparente) - α = 255 (opaco)
strokeWeight (valor)	Grosor del borde (en px) **OJO AL ESCRIBIR**
fill (r,g,b,α)	Color de relleno
noStroke ()	Sin borde
noFill ()	Sin relleno
background (r,g,b,α)	Color de fondo de la ventana

Para obtener cualquiera de los colores podemos utilizar abrir el menú principal en **Herramientas / Selector de colores.** Y podemos copiar los datos de color rgb o utilizar la opción “copiar”

Las propiedades de color deben **escribirse antes** de la función que describen

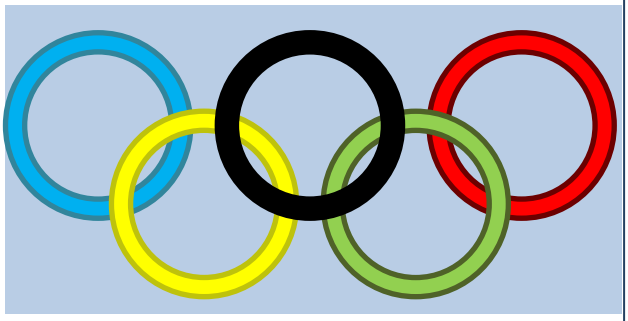
Ejemplo:





Ejercicio 2:

- Realiza dos programa que generen las siguientes imagenes
- Manda el archivo por mail



10. Estructura de Processing

```
1  int x=10;
2  int y=200;
3
4  void setup()
5  {
6    size(400, 400);
7    background(255, 0, 0);
8  }
9
10 void draw()
11 {
12   fill(#14B495);
13   ellipse(x, y, 40, 40);
14
15   x=x+20;
16 }
17
```

Hasta ahora hemos utilizado programas secuenciales, donde cada instrucción se ejecuta una detrás de otra y una sola vez.

Pero la estructura típica de un programa en Processing es con dos funciones:

- **Definición de datos y variables**
- **void setup () { }**
- **void draw () { }**

La función **setup** se ejecuta una sola vez al iniciar el programa. Por lo que por ejemplo la función size se va a ejecutar una sola vez al principio.

La función **draw** se repite constantemente y aquí es donde haremos el dibujo y todo el tiempo el programa va a estar redibujándolo. Por ejemplo, si dibujo una elipse (100,100,40,50), aunque no lo podamos ver, la instrucción draw, está redibujando la elipse constantemente una encima de otra. Esto permite dar movimiento a una imagen

11. Datos variables y operadores:

En todos los lenguajes de programación, es imprescindible el uso de datos. Para ello deberemos primero definir los tipos de datos que introduciremos en Processing según sea su naturaleza (entero, real, carácter, cadena de caracteres, etc).

- **Tipos de datos:**

Instrucción	Descripción	Ejemplo
int	números entero	Int a= 8;
float	números con decimales (los decimales se expresan con punto)	float b=20,35;
boolean	verdadero o falso	boolean c=true / false;
color	color que puede tener un pixel	color d= color(255,100,10); color d= #FF640A
char	un carácter o letra (se define con comillas simples)	char e= 'V';
String	cadena de caracteres o frase (se define con comillas dobles)	String f= "Es verdadero";

- **Variables**

Una variable es un espacio en memoria o cajón contenedor donde guardamos un valor. La variable tiene un nombre fijo y un valor que puede cambiar

Ejemplo, una variable puede ser la distancia de un robot móvil hacia la pared. El nombre de la variable, "dist_pared", y su valor lo expresaremos en centímetros. Si se acerca a la pared el valor de "dist_pared" disminuye. También puedo operar con esta variable y multiplicarla o sumarla cierto número, o la puedo comparar con otros valores, y eso me puede servir para tomar decisiones.

Las variables deben declararse antes de empezar el programa y normalmente se les asigna un valor de inicio.

```
int dist_pared = 40;        //declara una variable con el nombre "dist_pared" y la
                              inicia en valor 40 cm
println (dist_pared);       //imprime en consola el valor de la distancia a la pared
```

Los nombres de las variables:

- no pueden contener espacios, por lo que las palabras deben separarse por guiones bajos o utilizando mayúsculas al inicio de cada palabra (DistPared)
- no pueden contener caracteres especiales, excepto guión bajo (_)
- pueden contener números, pero no empezar por ellos

```
int a = 78;
float b = 13.46;
println (a*b);
```

```
size (300,500);
int x = 50;
int y = 60;
float tamano = 100.5;
rect (x,y,tamano,tamano);
```

Variables de sistema

Processing viene con una serie de variables internas que no tenemos que declarar. Algunas de estas variables son:

Instrucción	Descripción
width y **height**	ancho y alto de la ventana
keyPressed, keyReleased, mousePressed, mouseButton, mouseReleased	variables que reaccionan a las acciones del teclado o el ratón (presionar una tecla o el ratón o soltarlos)
Key	variable char que almacena la última tecla presionada en el teclado
mouseX y **mouseY**	posición del ratón en el eje X y eje Y
print ( ) y **println ( )**	imprimen mensajes en la consola de texto **print()** imprime el texto en una línea y se queda esperando por la siguiente orden print() o prinln() para imprimirla justo después, en la misma línea de la consola. En cambio **println()** imprime una línea completa y luego salta a la siguiente línea.

Operadores

Operadores matemáticos

Para operar debemos respetar la jerarquía matemática haciendo uso de los paréntesis “( )”,

+	suma	/	división
-	resta	++	incremento de 1
*	multiplicación	--	decremento de 1

Operadores de comparación:

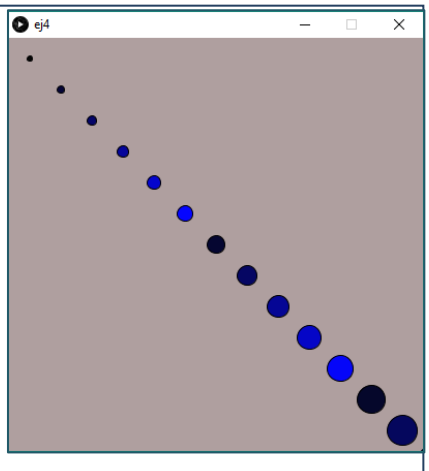
==	igual que	!=	diferente que
<	menor que	>	mayor que
<=	menor o igual que	>=	mayor o igual que

Operadores lógicos Y / O / NO

&&	Y lógico	Ej : (x > 0 && y < 5)
||	O lógico	Ej : (x > 0 || y > 0)
!	NO lógico	Ej : (!x > 0)

Ejercicio 3:

- Define dos variables a y b con valores iniciales de 20 en ambos casos, para indicar la posición del centro de las circunferencias
- Define una variable x con valor inicial de 5, para indicar el diámetro de las circunferencias
- Define una variable color c con valor inicial gris oscuro
- Define una ventana gris claro de tamaño 400x400 px
- Dibuja una circunferencia con origen a,b; de medida x de diámetro y de color c.
- Dibuja circunferencias desfasadas de la anterior 30 px en horizontal y vertical; de diámetro 2 px mayores que la anterior y de color que aumenta un valor de 50.
- Manda el archivo por mail

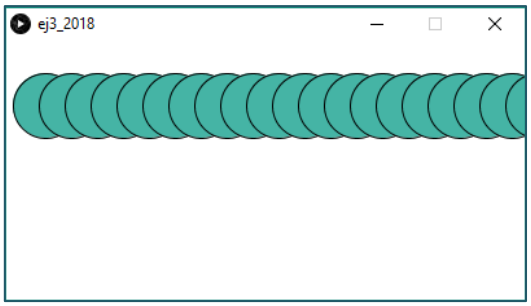


12. Dar movimiento a una imagen

Cambiando el valor de las coordenadas del dibujo mediante variables podemos hacer que las formas se desplacen por la pantalla.

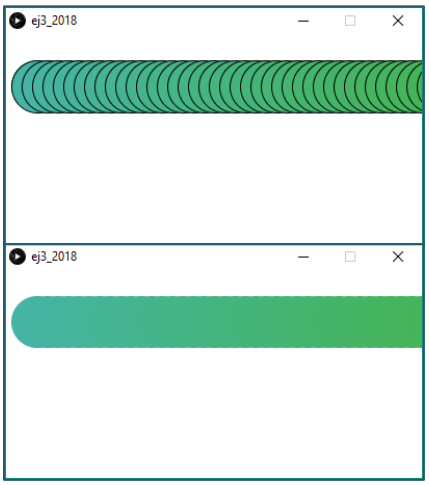
Ejemplo:

```
int a=30;
int b=50;

void setup()
{
  size(400, 200);
  background(255);
}

void draw()
{
  fill(#45B4A5);
  ellipse(a, b,50, 50);

  a=a+20;
 }
```



Podemos cambiar el color de la circunferencia a medida que avanza y eliminar o no el borde de la forma.

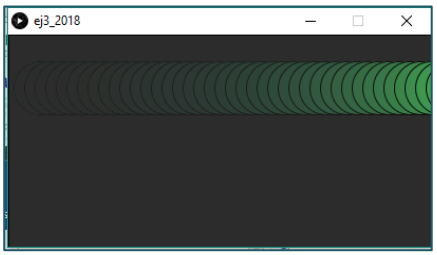
```
int b=50;
color c=#45B4A5;

void setup()
{
  size(400, 200);
  background(255);
}

void draw()
{
  noStroke();
  fill(c);
  ellipse(a, b,50, 50);

  a=a+10;
  c=c-2;
 }
```



Si no queremos que la imagen deje huella, colocaremos el color de fondo en el **void draw** para que se vaya redibujando antes de dibujar las siguientes circunferencias.
Y si queremos que la forma deje huella redibujaremos rectángulos de tamaño la ventana indicándolos color y transparencia

```
color c=#45B4A5;

void setup()
{
  size(400, 200);

}

void draw()
{
  background(#222726);

  fill(c);
  ellipse(a, b,50, 50);

  a=a+10;
  c=c-2;
 }
```

```
color c=#45B4A5;

void setup()
{
  size(400, 200);

}

void draw()
{
  fill(#222726,20);
  rect(0,0,400,200);

  fill(c);
  ellipse(a, b,50, 50);

  a=a+10;
  c=c-2;
 }
```



Para evitar que la forma se salga de la pantalla podemos utilizar una **estructura condicional**:

```
if (a>width)
   { a=0; }  // volvemos al principio
```

Ejercicio 4:

- Define dos variables a, b con valores iniciales de 50,50
- Define una ventana de tamaño 400x200 px de color rojo
- Dibuja una circunferencia de color azul, con origen a,y b de medida 40 px de diámetro
- La circunferencia debe avanzar en horizontal de 2 en 2px sin dejar huella y a la vez, el fondo de la ventana debe ir cambiando de tono (disminuyendo una unidad el tono rojo)
- Al finalizar la ventana la circunferencia debe empezar el recorrido y la ventana inicia otra vez en rojo
- Manda el archivo por mail

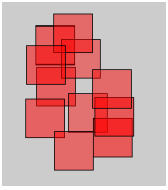
Ejercicio 4 EXTRA:

- EXTRA: cambia el tamaño de la ventana a 400x400. Haz que el avance de la circunferencia sea en diagonal. Y haz que la bola deje huella en su recorrido

13. Interacción con el ratón y el teclado

Instrucción	Descripción

mouseX y **mouseY**	posición del ratón en el eje X y eje Y.
	Se utilizan para dibujar al colocar el ratón en una posición
	Ejemplo: Dibujar un círculo siguiendo el cursor del ratón
	sketch_180226a
	`void setup(){`
	`  size(600,600);`
	`}`
	`void draw(){`
	`  fill(0,0,0,30);`
	`  rect(0,0,width,height);`
	`  fill(0,255,0);`
	`  ellipse(mouseX,mouseY, 50,50);`
	`  }`
	 

mousePressed,
mouseButton,
mouseReleased
 	
keyPressed,
keyReleased,
	variables que reaccionan a las acciones del ratón (**mouse)** o el teclado (**key**)
	Actúan al presionar una tecla o el ratón o al soltarlos
	Se suelen utilizar con la estructura condicional **if**
	`if (mousePressed)`
	`{`
	`fill(255,0,0,20);`
	`rect(mouseX,mouseY,50,50);`
	`}`
	 
	O se ejecutan en una **función independiente** debajo del void draw
	`void keyPressed()`
	`{`
	`fill(0,2550,0,20);`
	`rect(100,100,50,50);`
	`}`

Key **keyCode**	Variable que almacena la última tecla presionada en el teclado Variable que almacena la última tecla de control presionada en el teclado UP, DOWN, LEFT, RIGHT, ALT, CONTROL, SHIFT. Su uso debe combinarse con keyPressed para poder consultar la tecla presionada.

```
void keyPressed() {
    if (keyCode==LEFT) {
      fill(0,255,0));
      rect(100, 100, 100, b++);
    }
```

Ejercicio 5a:

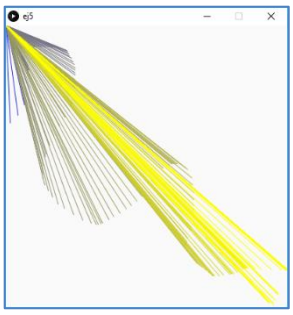
- Define una ventana de tamaño 500x500 px
- Realiza un programa que dibuje un circulo de 40px de diámetro en el centro de la ventana y que podamos modificar su diámetro mediante las flechas UP (aumentaría diámetro) y DOWN (disminuiría diámetro).
- Tomar la precaución de que no crezca más que el tamaño de la ventana.

Ejercicio 5b:

- Haz que el círculo avance hacia la derecha o la izquierda al pulsar las teclas RIGHT y LEFT del teclado
- El círculo no debe salir de la ventana
- Manda el archivo por mail

Ejercicio 5 extra:

- Crea un programa que dibuje líneas rectas de diferentes colores, entre el punto origen de coordenadas y el punto que tenga las coordenadas del ratón en una ventana de 400 x 400.
- Mueve el ratón y crea una figura
- Captura tu dibujo y manda la captura y el programa por mail



14. Sentencias de control: condicionales

- **if** (condición SI): Comprueba si una condición se cumple. La función se escribe entre llaves {}

```
if  (width > 500)  { x= -x ; }
```

- **if…else**: SI…..SINO: Ejecuta una u otra función dependiendo de si una condición se cumple o no

```
if (x > 120)    { println (verdadero); }
else   { println (falso); }
```

- **else if**:

Podemos **evaluar una serie de condiciones** anidando varias sentencias **else if**, siendo todas ellas, a su vez, mutuamente excluyentes:

```
if (x <= 0)          {point(20,20);}       //se ejecuta con x<0
else if (x < 100)    {ellipse(50,50,100,100);}     //se ejecuta con 0< x>100
else if (x <= 200)   {rect(50,50,100,100);}    //se ejecuta con 100< x>200
else                 {triangle(50,50,50,150,150,150;}   //se ejecuta con  x>200
```

- **if** anidados

Si se cumple una primera condición, **evaluar una segunda condición** anidando varias sentencias **if…**

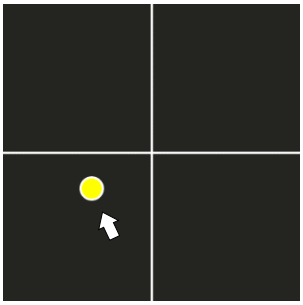
```
if (x <= 0)    {
    if (y>10)      {ellipse(50,50,100,100);} }    //se ejecuta con x<0 e y>10
else               {rect(50,50,100,100);}     { //se ejecuta en el resto de casos
```

- **switch…..case.** Se utiliza cuando tenemos 3 ó más funciones que ejecutar, según diferentes condiciones. (Funciona como la estructura if……else)

switch (letra) {	Al principio declaramos la variable char letra= 'N';
case 'A': funcion1 (); **break;**	
case 'B': funcion2 (); **break;**	Después podemos definir las funciones 1, 2 y 3 de forma independiente
case 'C': funcion3 (); **break; }**	void funcion1() { };
	void funcion2() { },
	void funcion3() { };

Ejercicio 6a:

- Define una ventana de tamaño 400x400 px y del color que quieras
- Dibuja una línea recta horizontal y otra vertical en medio de la pantalla (dividen la pantalla en 4 cuadrantes).
- Crea un programa que dibuje círculos siguiendo el movimiento del ratón y de diferente color según el cuadrante en el que se encuentre
- Manda el programa por mail



Ejercicio 6b:

- Realiza el ejercicio anterior, pero de forma que la pelota se mueva al pulsar las teclas UP, DOWN, LEFT, RIGHT
- El círculo no debe salir de la ventana
- Manda el programa por mail

15. Cambio de dirección en el movimiento de una imagen

Para conseguir que una forma con movimiento retroceda al llegar al final de la pantalla y cambie de dirección, necesitamos crear una **variable de avance** a la que multiplicaremos por un número negativo (-1) para que cambie de dirección.

```
Int avanceX=5;
```

Y al llegar al final de la pantalla le decimos que el avance sea negativo multiplicando por -1:

```
avanceX = avanceX * -1;
```

Ejercicio 7:

- Define una ventana de tamaño 600x200 px
- Realiza un programa que dibuje un circulo de 50px de diámetro en el margen izquierdo de la ventana y centrado respecto a la vertical.
- Crea una variable avance que empiece en 5
- Haz que el círculo avance dejando huella hacia la derecha aumentando cada vez el avance (de 5 en 5) hasta tocar el final de la ventana
- Haz que luego retroceda hasta tocar el principio de la ventana, multiplicando el avance por -1
- Manda el archivo por mail

16. Bucles

- **for** (DURANTE): Es un bucle que repite un bloque encerrado entre llaves un número de veces mientras una condición se cumple. Utiliza una variable para contar el número de veces que se han repetido las órdenes.

  Lleva 3 comandos:

  for (declara la variable contadora y la inicia; condición para que se repita el bucle; incremento a cada paso del bucle)

```
for (int x = 0; x<400; x =x+5)
{ line(30,x,80,x); } // Hace líneas paralelas desde la posición 0
                     hasta la 400 de x
```

  Cada vez que se va a repetir el bucle, se revisa la condición, si es cierta, el bloque de funciones se ejecuta (en este caso, se suma), y la condición vuelve a ser comprobada de nuevo. Si la condición es falsa, el bucle termina

- **while** (MIENTRAS): Es un bucle que se repite infinitamente mientras la expresión de dentro del paréntesis sea cierta.

```
Int x= 0;
while (x < 200)
   { line(30,x,80,x);
     X=x+5; }     // Hace líneas paralelas desde la posición 0 hasta
                     la 200 de x
```

- **do…while** (HACER….MIENTRAS**:** Trabaja de la misma manera que el bucle *while*, con la excepción de que la condición se comprueba al final del bucle, por lo que este bucle se ejecuta "siempre" al menos una vez.

```
Int x = 0;
do { line(30,x,80,x);
      x= x+5; }
while (x < 200)
```

- **break** (ROMPER**):** Es usado para salir de los bucles *do*, *for*, o *while*, pasando por alto la condición normal del bucle.

```
for (i = 0; i < 10; i++)
{ Console.WriteLine("valor de i: {0}", i);
if (i > 4)   {break; }
```

17. Generar valores aleatorios: función random

La función **random (x,y)** genera valores aleatorios entre dos límites x e y. Los valores aleatorios no funcionan con variables de tipo **int**

Prueba el siguiente programa:

```
void setup(){
    size(400,500);
}

void draw(){
    ellipse(random(0,400), random(0,500), 30, 30);
}
```

```
float x=0;
float y=0;

void setup(){
    size(400,500);
}

void draw(){
  x=random(0,400);
  y=random(100,300);

  ellipse(x, y, 40, 40);
}
```

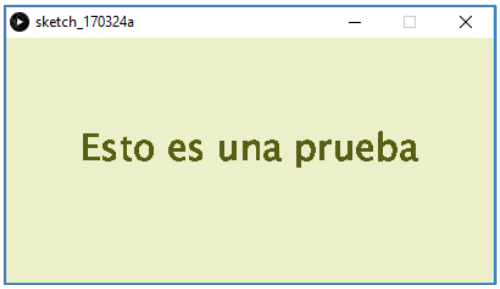
Ejercicio 8:

- Define una ventana de tamaño 400x500 px y de color negro
- Dibuja círculos en posición aleatoria, de color aleatorio y de tamaño aleatorio entre 10 y 50, sin dejar huella.
- Manda el programa por mail

18. Escribir texto

Instrucción	Descripción	Ejemplo
textSize (n)	Tamaño de letra expresada en puntos	textSize (12);
fill (R,G,B)	Color de texto	fill (23,45,200);
textAlign (modo)	Alineación del texto: **RIGHT, LEFT, CENTER**	textAlign (CENTER);
text (“datos”, x, y)	Escribe los datos en la posición x,y	text (“hola”, 20,30);

```
void setup()
{
  size(400,200);
  background(#EBF0CB);
}

void draw()
{
 textSize (30);
 fill(#565F14);
 textAlign(CENTER);
 text("Esto es una prueba", width/2,height/2);
}
```



Ejercicio 9:

- Define una ventana de tamaño 200x400 px y del color que quieras
- a) Crea un programa que reproduzca tu nombre de forma similar a la imagen
  - Aumenta el tamaño de letra y color de letra, progresivamente
- b)I nserta un **bucle for** que establezca el número de repeticiones en 15 (todas las variables que cambian deben insertarse dentro del bucle for)
- Manda los dos programas por mail

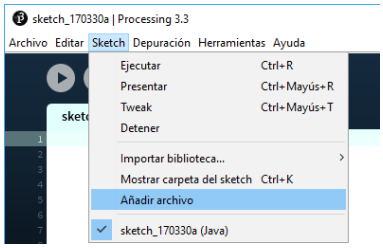
Para incluir una variable dentro de un texto debemos indicar la variable de la siguiente manera: **“+variable+”**

```
Int n=8;
text ("Puntuación="+n+" ", 40, 40);
```

19. Insertar imágenes

Primero debemos guardar la imagen en la carpeta de nuestro proyecto. Para ello debemos pulsar **Sketch/ Añadir archivo**. La imagen ya queda dentro de la carpeta data de nuestro proyecto.

O creamos la carpeta data dentro de nuestro proyecto e introducimos dentro las imágenes



- Primero debemos declarar las variables como imágenes con la expresión **PImage** y damos un nombre a nuestra variable imagen.

  PImage fondo;

- Luego en el **void setup** cargamos nuestra imagen: **loadImage** (nombre imagen)

  fondo = **loadImage** (“cielo.jpg”);

- En el **void draw** incluimos la función **image** (nombre variable, posición x, posición y, dimensión x, dimensión y)

  image (fondo,0,0);

  También podemos cargarla en el void setup si la imagen no se va a modificar durante el programa.

Si no ponemos dimensiones, la imagen se carga a su tamaño real.

ejemplo_imagen

```
1 PImage imagen1;
2
3 void setup() {
4   size (720,390);
5   imagen1 = loadImage("Universo-1.jpg");
6 }
7 void draw() {
8   image (imagen1,10,10,700,370);
9 }
```

Ejercicio 10a:

- Busca una imagen de fondo de un cielo nocturno y de una nave espacial.
- Abre un archivo nuevo llamado “juego_nave_”espacial” e introduce las imágenes en la carpeta data
- Define una ventana del tamaño de tu fondo (aproximadamente 1000x 600)
- Coloca el fondo y la nave espacial.
- Haz que la nave se mueva hacia la derecha o la izquierda al pulsar las teclas RIGHT y LEFT del teclado
- La nave no debe salir de la ventana
- Manda el programa por mail

Ejercicio 10b:

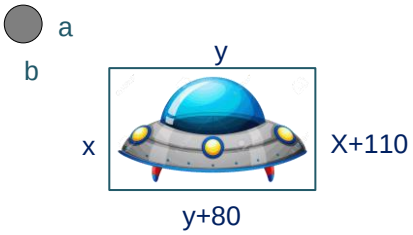
- Crea las variables a y b, para definir la posición de una bola.
- Crea una bola blanca de diámetro 30 y colocala en la parte superior izquierda de la ventana
- Crea las variables vela y velb, que van a definir la velocidad a la que cae la bola. Inicia las variables con el valor 3.
- Haz que la bola caiga utilizando el siguiente código

  ellipse(width/2+a, 10+b, 30,30);
  a=a+vela;
  b=b+velb;
- Haz que la bola rebote en los bordes de la ventana
- Manda el programa por mail

Ejercicio 10c:

Vamos a comparar las posiciones de la bola y de la nave para que cuando choquen la bola cambie la trayectoria.
Chocarán cuando se cumpla:

a>x
a<x+110
b>y



- Programa el cambio de dirección cuando choquen utilizando el siguiente código

```
if ( (a >x) && (a<x+110) && (b >y) )
{
   velb=velb*-1;
 }
```

- Manda el programa por mail

Ejercicio 10d:

- Crea la variable puntos. E iniciala en 0
- Incluye un texto “puntuación” donde irá la variable puntos
- Suma puntos cuando la nave choque con la bola
- Resta puntos cuando la bola caiga y no la pare la nave
- Manda el programa por mail