Hola Mundo Con Processing

Rocío Abascal Mena
Erick López Ornelas
Sergio Zepeda Hernández

Obra ganadora Concurso 2014 para publicación de libros de texto y materiales de apoyo a la Docencia

10 años
2005 – 2015
Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA
Unidad Cuajimalpa

Rocío Abascal Mena
Erick López Ornelas
Sergio Zepeda Hernández

Hola Mundo con Processing

Esta investigación fue dictaminada por pares académicos

Clasificación Dewey: 005.1 A23
Clasificación LC: QA76.73.P75 A23
Abascal Mena, Rocío
Hola mundo con processing / Rocío Abascal Mena, Erick López Ornelas, Sergio Zepeda Hernández. -- México : UAM, Unidad Cuajimalpa, c2015.
90 p. : il. col. ; 24 cm. -- (Una década de la Unidad Cuajimalpa de la Universidad Autónoma Metropolitana)
ISBN de la Colección Una Década: 978-607-28-0452-4
ISBN de este libro: 978-607-28-0475-3
1. Processing (Lenguaje de programación para computadora) – Libros de texto 2. Universidad Autónoma Metropolitana – Unidad Cuajimalpa – Planes de estudio 3. Programación de computadoras 4. Planes de estudio universitario - México
I. López Ornelas, Erick, coaut. II. Zepeda Hernández, Sergio, coaut.

Universidad Autónoma Metropolitana Unidad Cuajimalpa
.Avenida Vasco de Quiroga 4871,
Col. Santa Fe Cuajimalpa. Delegación Cuajimalpa de Morelos,
C.P. 05348, México D.F. (Tel.: 5814 6500)
www.cua.uam.mx

ISBN DE LA COLECCIÓN UNA DÉCADA: 978-607-28-0452-4
ISBN DE ESTE LIBRO: 978-607-28-0475-3

Diseño de portada: Ricardo López Gómez.
Formación y edición: Juan Carlos Rosas Ramírez.

CONTENIDO

Introducción

Hasta hace algunos años, el aprender a programar solo era accesible para una élite que pasaba mucho tiempo frente a una computadora intentando escribir algunas líneas de código. Sin embargo, ahora aparecen cada vez más herramientas que inducen a la programación desde una temprana edad. Dichas herramientas ofrecen un entorno amigable y diversas alternativas de aprendizaje de los elementos básicos de los lenguajes de alto nivel: variables, estructuras de control, sentencias, funciones, condicionales, operadores, etc. A su vez, el aprendizaje de fundamentos de programación a partir de este tipo de herramientas facilita el desarrollo de diversas habilidades multidisciplinares que se ponen en marcha cuando se elaboran pequeños programas encaminados, por ejemplo, a la resolución de problemas, la creación de juegos sencillos e incluso la generación de escenarios complejos en 3D.

El doctor Mitchel Resnick, director del Grupo Kindergarten Lifelong en el MIT Media Lab, del Massachusetts Institute of Technology, sostiene que en la actualidad saber programar es como saber leer y escribir. También es posible compararlo con aprender otro idioma, ya que cada lenguaje de programación que aprendamos nos sirve de la misma manera que nos sirve conocer otra lengua. En un artículo, Resnick asegura que existen razones más profundas y generales para aprender a programar:

> en el proceso de aprender a programar, las personas aprenden muchas otras cosas. No están simplemente aprendiendo a programar, están programando para aprender; pues además de comprender ideas matemáticas y computacionales, tales como variables y condicionales, simultáneamente están aprendiendo estrategias para solucionar problemas, diseñar proyectos y comunicar ideas. Esas habilidades son útiles no solo para los científicos de la computación, sino para todas las personas sin distingos de

edad, proveniencia, intereses u ocupación (Recuperado de https://www.edsurge.com/n/2013-05-08-learn-to-code-code-to-learn. Consultado el 29 de septiembre de 2014).

En la Unidad Cuajimalpa de la Universidad Autónoma Metropolitana, se imparte desde 2009 la Unidad de Enseñanza-Aprendizaje (UEA) Fundamentos de programación estructurada, que utiliza el lenguaje de programación Processing (http://processing.org/). Este lenguaje fue desarrollado en 2001 en el MIT por Casey Reas y Ben Fry. Processing está basado en Java pero debido a su simplicidad permite visualizar gráficamente los resultados de un ejercicio y concentrarse en los fundamentos propios de la programación más que en la sintaxis del lenguaje.

Al usar Processing para el aprendizaje de los fundamentos de programación se aprecia, desde los primeros días de clase, un entusiasmo y motivación diferentes en comparación con alumnos que se enfrentan a la programación, por primera vez, con lenguajes como C, Pascal, entre otros. Incluso, el nivel de reprobación es menor en clases donde se utiliza Processing.

En este material mostramos, paso a paso, el curso que se imparte en la UEA Fundamentos de programación estructurada, a la vez que se presentan ejemplos de programas realizados por alumnos de distintas generaciones.

Consideramos que el presente material es de gran valor y servirá para motivar a futuras generaciones que se convertirán en talentosos programadores visuales.

Relación del contenido con el programa de estudios de la UEA Fundamentos de programación estructurada

La UEA Fundamentos de Programación Estructurada (450107), que se imparte en la Licenciatura de Ciencias de la Comunicación y en la Licenciatura de Diseño, responde a la necesidad de ampliar las capacidades del alumno, debido a que el aprender a programar permite estructurar las ideas de una manera diferente con el fin de lograr un objetivo.

Esta UEA tiene como objetivo general que al final del curso "el alumno sea capaz de comprender y aplicar los conceptos básicos de programación estructurada para la solución de problemas". Por ello, el curso está diseñado para que el alumno resuelva problemas desde el primer día a partir de la descomposición de los mismos en problemas más pequeños. Respecto a los objetivos específicos, estos incluyen:

1. *Utilizar adecuadamente la lógica matemática aplicada a la metodología de la programación.*
   La lógica matemática es una rama fundamental de las matemáticas que establece el valor de verdad de las proposiciones y permite construir el razonamiento matemático. En el curso Fundamentos de Programación Estructurada, se utilizan proposiciones simples y complejas que pueden tomar valores verdaderos o falsos. Las posibilidades de que una proposición pueda ser verdadera se ordenan en una "tabla de verdad", la cual refleja gráficamente las posibilidades de obtener verdadero o falso. Los alumnos utilizan las tablas de verdad y el uso de la lógica matemática en el Capítulo 6: Condicionales.
2. *Utilizar pseudocódigos y diagramas para simplificar un problema de programación.*
   El Capítulo 1: Introducción a la programación presenta el uso de pseudocódigos y diagramas de flujo para la solución de problemas. Sin embargo,

el pseudocódigo se emplea a lo largo de todo el curso como un medio para esclarecer ideas y entender el problema antes de programar con Processing.

3. *Conocer y aplicar lo imprescindible de la programación estructurada para realizar programas sencillos*.
   En el curso de Fundamentos de Programación Estructurada, se conoce y aplica lo imprescindible de la programación estructurada. Además, se considera que las variables, las condicionales y las estructuras de control son imprescindibles en cualquier lenguaje de programación. Estos temas se abordan en el capítulo 4, variables); 5 condicionales, y en el 6, Estructuras de iteración.
4. *Adquirir las habilidades básicas en programación estructurada usando tipos de datos, estructuras de control y estructuras complejas de datos*.
   En el cuarto objetivo de la UEA Fundamentos de Programación Estructurada, se hace énfasis en la adquisición de habilidades básicas de programación a partir del uso de tipos de datos, estructuras de control y estructuras complejas de datos. Estas habilidades consisten en la capacidad de plasmar la solución de un problema en instrucciones.

El contenido de este documento material se relaciona por completo con la UEA Fundamentos de Programación Estructurada, ya que se elaboró con base en la impartición de dicho curso desde el año 2009. Los temas han evolucionado y se han complementando a partir de los resultados obtenidos con las distintas generaciones. Se ha puesto especial atención en la práctica y en la aplicación de ejercicios muy variados, que se vuelven más complejos para que el alumno conozca y aplique lo que se mostró en clase.

El presente material se basa en la experiencia de los profesores que han impartido esta UEA tanto en la Licenciatura de Ciencias de la Comunicación como en la Licenciatura en Diseño. Asimismo, se muestran ejercicios realizados por los alumnos que pueden servir de base para futuras generaciones.

Descripción de la importancia de los conocimientos a adquirir, así como de las habilidades y actitudes a desarrollar

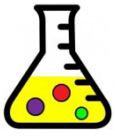
Los conocimientos que se adquieran en este material permitirán al alumno su aplicación en cualquier lenguaje de programación. Aun cuando se utiliza Processing para la adquisición de los conocimientos, estos no están sujetos a un único lenguaje de programación. A lo largo de los años, hemos aprendido a proporcionar al alumno una serie de habilidades y actitudes que le permiten entender la programación de una manera agradable. Los alumnos descubren su habilidad por la programación cuando se dan cuenta de que pueden lograr lo que se plantean.

Entre las habilidades que se desarrollan están:

1. *Abstracción*. Habilidad para identificar las características y el comportamiento de un objeto.
2. *Pensamiento creativo*. Habilidad para crear, identificar, plantear y lograr una solución divergente de un problema. Los alumnos aprenden que no hay una única solución para un problema y que se necestia emplear la creatividad para resolverlo. El pensamiento creativo también es estimulado con programas libres que los motivian a desarrollar lo aprendido. Estos programas estimulan la competencia entre los alumnos.
3. *Identificación de errores de programación e implementación de soluciones*.
4 *Pensamiento ordenado*. Una de las grandes habilidades que se logran al finalizar el curso es el poder descomponer un gran problema en subproblemas. De este modo, el alumno concibe que todo está basado en una serie de pasos que pueden ser programables. Al finalizar el trimestre, el alumno logra un pensamiento organizado y documenta las soluciones de los problemas de manera ordenada.

Iconografía utilizada

A lo largo del material, se utilizan un conjunto de íconos que muestran cada una de las tareas a realizar. La lista de los íconos utilizados es la siguiente:

Objetivo	
Actividades de aprendizaje	
Ejemplos	
Ejercicios	
Actividad integradora	
Experiencias previas	

Capítulo 1. Introducción a la programación

"Primero resuelve el problema. Entonces, escribe el código".
John Johnson

Objetivo

- El alumno entenderá qué es programar con base en el concepto de algoritmo y a través de realizar un conjunto de *Diagramas de Flujo* con datos básicos. Además, se introducirá al lenguaje de programación Processing.

Actividades de aprendizaje

- El alumno identificará el concepto de algoritmo mediante representaciones de su vida cotidiana.
- Aplicará el concepto de *Diagramas de Flujo de Datos* para representar algoritmos cotidianos.
- Interactuará con la interfaz inicial de Processing.

El algoritmo y sus características

Programar es escribir algoritmos. Al escribir líneas de código que posteriormente generarán un programa de cómputo, lo que estamos haciendo es traducir un algoritmo. Pero, ¿qué es entonces un algoritmo? Un algoritmo es simplemente un conjunto de pasos ordenados que nos ayudarán a resolver cualquier problema.

En nuestra vida cotidiana, realizamos un conjunto de acciones (muchas veces sin darnos cuenta). ¿Cómo llegamos a la universidad desde nuestra casa? ¿Qué pasos se deben de seguir para la inscripción a un curso de la universidad? ¿Qué se debe de hacer para comer en la cafetería? ¿Cómo se obtiene un libro de la biblioteca? ¿Cómo se envía a imprimir en la sala de cómputo? Estas acciones se realizan de manera ordenada para lograr un objetivo. Podemos decir que estas acciones son algoritmos, una serie de instrucciones que debemos de seguir para lograr un fin deseado. Prácticamente en todas las

tareas que realizamos diariamente estamos, sin darnos cuenta, ejecutando algoritmos.

Ejemplo 1

¿Cuáles son los pasos que debemos seguir para comer en la cafetería?
La respuesta es:

Ir a la caja para comprar un boleto de la cafetería
Desplazarse a la cafetería
Hacer la fila para poder entrar
Seleccionar alimentos para comer
Seleccionar postre y agua
Entregar el boleto de comida
Buscar un lugar para sentarse
Comer los alimentos seleccionados

El Ejemplo 1 muestra un algoritmo muy sencillo, cuya finalidad es poder comer en la cafetería. Como en el ejemplo anterior, estamos constantemente ejecutando algoritmos para poder lograr todos nuestros objetivos, aunque muchas veces lo hacemos de manera inconsciente.

Un algoritmo debe de tener tres características importantes:

1. Un algoritmo debe ser *definido*. Esto quiere decir que el algoritmo debe de tener un orden en la realización de cada uno de los pasos. Si alteramos este orden, el resultado no será el planeado.
2. Un algoritmo debe de ser *preciso*. Si un algoritmo se ejecuta dos o más veces siempre deberá proporcionar el mismo resultado.
3. Un algoritmo debe ser *finito*. Esto quiere decir que el número de pasos a ejecutar deben de ser finitos. En otras palabras, el algoritmo debe siempre de terminar en un momento dado.

Si nuestro algoritmo cumple con estas características, entonces podemos decir que el algoritmo está bien construido.

Un algoritmo es un poco más complicado que una simple secuencia de pasos, existen dos elementos adicionales que debemos de tomar en cuenta al diseñar un algoritmo.

El primero es la *decisión*: al plantear el algoritmo, debemos de tomar en cuenta y hacer una previsión acerca de las diversas eventualidades que

podrían afectar la continuidad del mismo. No solo se trata de hacer la previsión, sino de dar una solución adecuada para lograr el resultado esperado al finalizar el algoritmo.

El segundo es la *iteración*: un conjunto de pasos pueden repetirse un número determinado de veces o mientras una condición no se cumpla. Este elemento también debe estar diseñado para que el algoritmo brinde el resultado esperado.

A continuación se muestra un ejemplo que incluye estos dos elementos adicionales, a los que llamaremos estructuras de control.

Ejemplo 2

¿Cuáles son los pasos que debemos de seguir para comer en la cafetería?
La respuesta es:

1. Ir a la caja para comprar un boleto de la cafetería
   a. Si no hay boletos, entonces pedir un boleto prestado
2. Desplazarse a la cafetería
3. Formarme en la fila para poder entrar
   b. Mientras estoy en la fila, entonces hago la tarea
4. Seleccionar alimentos para comer
5. Seleccionar postre y agua
6. Entregar boleto de comida
7. Buscar un lugar para sentarse
8. Si no hay lugar, entonces esperar a que se libere una mesa
   c. Comer los alimentos seleccionados

Estas estructuras de control adicionales (la decisión y la iteración) harán que el algoritmo sea flexible y completo.

Ejercicio 1 propuesto

Diseña los algoritmos correspondientes a las siguientes acciones y agrega estructuras de control para obtener el resultado deseado:

- Pasos para llegar a la universidad desde tu casa
- Pasos para realizar un pastel de elote
- Pasos para hacer una carne asada
- Pasos para cambiar la llanta de un auto
- Pasos para inscribirse en la universidad
- Pasos para obtener la credencial del INE
- Pasos para obtener un pasaporte

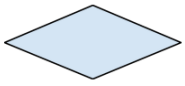
EL DIAGRAMA DE FLUJO DE DATOS

Los diagramas de flujo son una representación gráfica de un algoritmo o proceso. Un diagrama de flujo tiene generalmente un único punto de inicio y un único punto final. Además, cuenta con un conjunto de elementos gráficos que nos ayudarán a representar cada una de las acciones o procesos de un algoritmo y, sobre todo, a verificar el flujo de la información.

Existe un conjunto muy variado de elementos gráficos. Sin embargo, para nuestros propósitos, utilizaremos únicamente los siguientes elementos:

Símbolo utilizado para definir el inicio y fin del diagrama

Símbolo que nos representa todos los procesos o acciones



Símbolo de decisión que nos sirve para generar una bifurcación en el flujo de la información

Símbolo que nos sirve para unir todos los elementos generados y poder seguir el flujo de la información

Ventajas de usar una representación basada en Diagramas de Flujo de Datos:

- Favorece la comprensión del proceso al mostrarlo como un dibujo. Un buen diagrama de flujo reemplaza varias páginas de texto.
- Permite identificar los problemas y las oportunidades para mejorar el proceso. Se identifican los pasos, los flujos de los procesos, los cuellos de botella y los puntos de decisión.

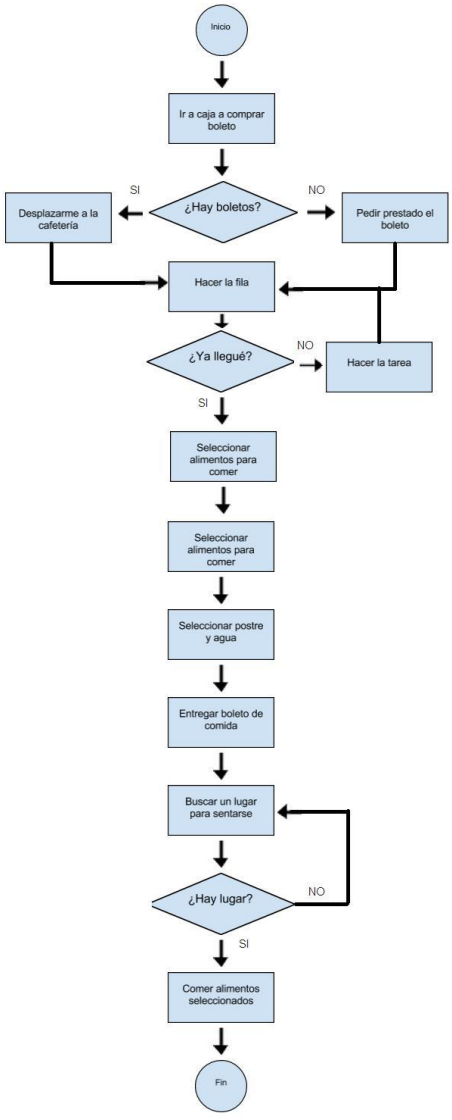
La creación de diagramas de flujo es simple, sin embargo, para su construcción debemos de seguir algunas reglas:

- Los diagramas de flujo deben de escribirse de arriba hacia abajo, y/o de izquierda a derecha.
- Los símbolos se unen con líneas, las cuales tienen, en la punta, una flecha que indica la dirección en la que fluye la información de los procesos. Se deben de utilizar solamente líneas de flujo horizontal o verticales, nunca diagonales.
- Se debe de evitar el cruce de líneas.
- No deben de quedar líneas de flujo sin conectar.

- Todo texto escrito dentro de un símbolo debe de ser legible, es decir, preciso.
- Solo los símbolos de decisión pueden y deben de tener más de una línea de flujo de salida.

A continuación se muestra un ejemplo de uso de estos diagramas.

Ejemplo 3



Ejercicio 2 propuesto

Diseña los diagramas de flujo correspondientes para los algoritmos que se especifican a continuación. Agrega estructuras de control para obtener el resultado deseado:

Pasos que se deben de realizar para llegar a la universidad
Pasos para hacer un pastel de elote
Pasos para hacer una carne asada
Pasos para cambiar una llanta de un auto
Pasos para inscribirse en la universidad
Pasos para obtener la credencial del INE
Pasos para obtener un pasaporte

EL ARTE DE PROGRAMAR

La computadora es una máquina con extraordinarias capacidades (elaboración de cálculos matemáticos, muestra de gráficos e imágenes, procesamiento y almacenamiento datos, etc.) pero incapaz de hacer algo por sí misma. La potencialidad está a la espera de que una persona (nosotros) le saque provecho y, para ello, lo único que se necesita es proporcionarle instrucciones.

Programar es definir instrucciones para ser ejecutadas por una computadora (programa). El objetivo de programar es invariablemente resolver un problema.

El poder programar es entonces el poder comunicarse con una máquina, darle instrucciones para que ejecute las tareas que se le piden. Estos programas se basan en algoritmos y en los diagramas de flujo de datos, los cuales nos ayudarán a solucionar algún problema. Si un programa es la traducción de un algoritmo, entonces también tendrá que heredar sus características básicas: ser preciso, estar bien definido y ser finito.

El programa es la traducción de un algoritmo en un conjunto de instrucciones o líneas de código que seguirán una sintaxis determinada, que el lenguaje de programación podrá entender y posteriormente ejecutar.

Programar es un arte porque es un medio de creación de cada uno de los individuos que interactúan con la computadora. Hay programadores expertos o novatos, pero lo importante es que la creación de un programa es única y los programadores podrán llegar a un mismo resultado de manera muy distinta. Esta es una característica única del arte de programar.

Los lenguajes de programación

Un lenguaje de programación es un lenguaje formal diseñado para expresar procesos que pueden ser llevados a cabo por las computadoras. Está formado por un conjunto de símbolos, así como de reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Las computadoras pueden ser programadas en muchos y muy distintos lenguajes de programación, por ejemplo C, Pascal, MatLab, Python, Fortran, Java, Lisp, R, entre muchos otros. Nosotros utilizaremos *Processing*, el cual está basado en Java y permite obtener resultados visuales de manera natural, lo que facilita el entendimiento de la Programación Estructurada.

Iniciación a Processing

Processing es un lenguaje de programación open source y un ambiente para la gente que quiere programar imágenes, animación y sonido. Es utilizado por estudiantes, artistas, diseñadores, arquitectos, investigadores e interesados en aprender, prototipar y producir. Ha sido creado para enseñar los fundamentos de la programación dentro de un contexto visual.

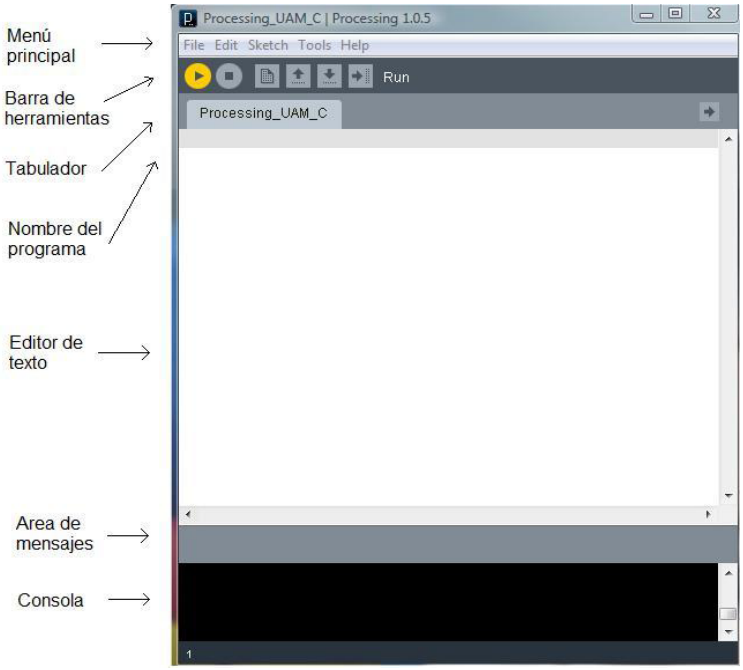
Todos los recursos sobre Processing se encuentran en la página web: http://www.processing.org, misma que incluye ejemplos, referencias y tutoriales, así como las librerías con las que se cuenta, blogs y, además, se puede descargar de manera gratuita el software. La versión más actual de Processing también está disponible para dispositivos móviles. Se debe de seleccionar la versión que se necesite en función del sistema operativo (Windows, Linux o Mac OS) con que se cuente.

- Para la versión de Windows, se deberá descargar un archivo .zip. Se debe de hacer clic en este y arrastrar la carpeta que contiene a cualquier sitio del disco duro. Es importante extraer la carpeta del .zip. Después se debe de hacer doble clic en processing.exe para empezar.
- Para la versión de Mac OS X, también se debe de descargar un .zip. Se hace doble clic sobre este y se arrastra el ícono de Processing a la carpeta de Aplicaciones. Después, se debe de dar doble clic en el ícono de Processing para empezar.
- Para la versión de Linux se descarga un archivo tar.gz, que es familiar para la mayoría de los usuarios de Linux. Posteriormente, se debe descargar el archivo al directorio Home, abrir la terminal de Windows y escribir:

tar xvfz processing-xxxx.tgz. Se debe reemplazar las xxxx por el nombre del archivo, en función de su versión. Otra opción es descomprimirlo y ejecutar el archivo *Processing*.

Una característica importante de Processing es que cuenta con una interfaz simple, de manera contraria a otros entornos de desarrollo. Además, permite concentrarse exclusivamente en el conjunto de instrucciones y en la observación de resultados visuales de manera rápida.

Cuando se ejecuta el archivo Processing.exe, se presenta la siguiente interfaz.



Los elementos con los que cuenta son simples: un menú principal con las opciones File, Edit, Sketch, Tools y Help. También cuenta con una barra de herramientas con seis opciones, mediante las cuales se puede tener todo el control sobre el programa. Estas opciones son:

- Ejecutar el programa 
- Detener la ejecución del programa 
- Crear un nuevo programa 

- Cargar un programa existente 
- Guardar el programa creado  . Por defecto, los programas son guardados en el *sketchbook*, una carpeta que almacena los programas. Dentro de la carpeta se encuentra el nombre del programa, guardado con la extensión .pde, que significa Processing Development Environment.
- *Exportar el programa*  . Esta opción permitirá compilar el código en una aplicación para distintos sistemas operativos (Mac OS, Windows y/o Linux). Esto es una manera fácil de crear versiones ejecutables de los proyectos. Por ejemplo, visualizar el programa en una página web porque genera los archivos HTML.

En el tabulador se mostrará el o los programas que estén activos. Posteriormente, se encuentra el editor de texto. En esta área de trabajo, escribiremos el conjunto de instrucciones, es decir, nuestro programa. En el área de mensajes, se mostrarán los posibles errores cometidos durante la ejecución del programa. Finalmente, en la consola es posible visualizar la ejecución del programa y tener una interacción con la ejecución del programa.

Al ejecutar cualquier programa  , siempre se desplegará una ventana. En esta ventana se podrá observar el resultado gráfico de la ejecución del programa. El siguiente ejemplo muestra, por el momento, una ventana vacía, debido a que no existe un elemento gráfico a desplegar.

Aprender a programar en Processing implica explorar mucho código: ejecutarlo, cambiarlo y mejorarlo hasta obtener nuevas cosas. Con esta idea

en mente, la descarga del software de Processing contiene un conjunto de ejemplos que muestran las diferentes características del software. Para abrir un ejemplo se debe de seleccionar ejemplos desde el menú *File* o hacer clic en el ícono *Open* del PDE. Los ejemplos están agrupados en categorías basadas según su función.

Al escribir el programa, se encontrarán palabras con distinto color (normalmente naranjas), esto significa que es una palabra reservada por Processing. Se puede entonces seleccionar y verificar la referencia para ver de qué se trata.

La *Referencia* que se encuentra en http://processing.org explica todos los elementos de código con una descripción y ejemplos. Los programas de ejemplo que se encuentran en la referencia son generalmente cortos y fáciles de seguir. Por ello, es importante consultar siempre http://processing.org para cualquier duda con respecto a una instrucción.

En el siguiente capítulo, empezaremos a programar los primeros elementos gráficos básicos, los cuales se complementarán con una explicación del uso del color.

Actividad integradora

Un estudiante de la UAM C está inscrito en la UEA Programación estructurada. Al llegar a casa, su papá le pregunta: "¿Cómo vas a hacer para obtener la mejor calificación en la UEA?"

Esta respuesta requiere una explicación detallada del estudiante: ¿cómo lo haría? ¿Cómo lo representaría de manera gráfica para que se entendiera mejor? ¿Podría explicar qué herramienta utilizaría para lograrlo? ¿Podría representar la serie de pasos que va a realizar para estudiar?

Capítulo 2. Dibujo

"Los mejores programadores no son solo marginalmente mejores que los buenos. Se trata de un orden de magnitud mayor, medida por cualquier estándar: creatividad conceptual, velocidad, ingenio o habilidad para solucionar problemas".
Randall E. Stross

Objetivo

- El alumno entenderá el concepto de pixel e identificará las formas geométricas básicas para representar cualquier forma, además, aprenderá a utilizar los elementos de color y su codificación utilizando Processing.

Actividades de aprendizaje

- El alumno explorará las funciones utilizadas para plasmar las formas básicas, así como los parámetros utilizados.
- Interactuará con el editor de texto de Processing para ejecutar sus programas usando funciones nuevas (`size()`, `fill()`, `noFill()`, `background()`).
- Utilizará la herramienta de *color selector* para identificar el uso de colores con sus representaciones RGB, HSB y codificación hexadecimal.

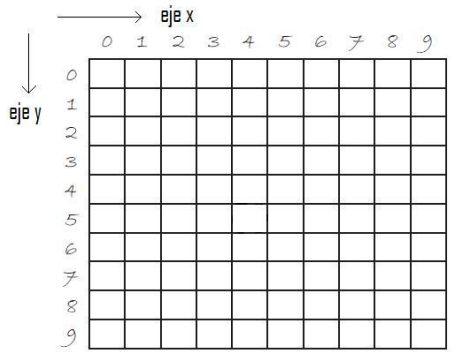
El pixel y formas básicas

El *pixel*, término muy utilizado en computación, se refiere al elemento más pequeño de los que componen una imagen digital. Empezamos esta iniciación a Processing con esta definición porque es el elemento visual básico que nosotros podemos representar en Processing.

De hecho, el monitor de la computadora está compuesto por un conjunto de pixeles que están organizados en una retícula rectangular, donde

la mínima información puede ser utilizada por Processing para representar elementos visuales.

Para poder ubicar todos los elementos en dicha retícula, las coordenadas iniciales (0,0) se ubicarán en la parte superior izquierda. Las coordenadas en el eje X aumentarán hacia la derecha y las coordenadas en el eje Y aumentarán hacia abajo.

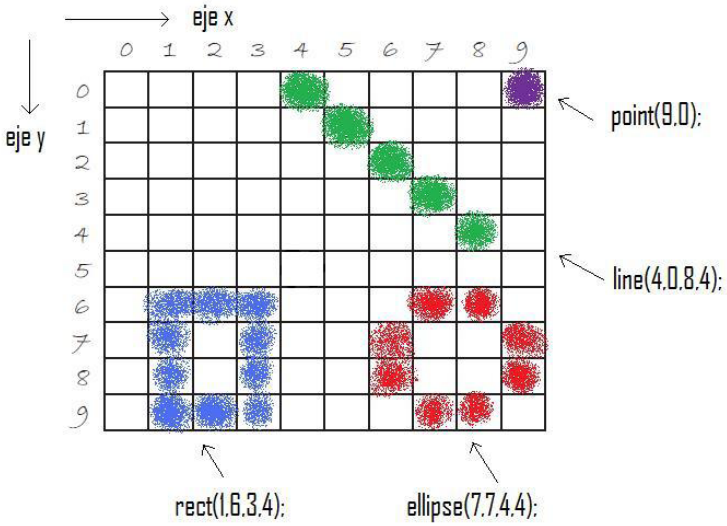


En esta retícula o espacio de trabajo, Processing representará todos los elementos visuales. Esta podrá ser cambiada de tamaño según lo que se necesite representar. Processing podrá representar, entonces, las formas básicas conocidas como funciones, las cuales ya están implementadas en Processing.

Las funciones básicas que utiliza Processing son las siguientes:

```
point(x,y);
line(x1, y1, x2, y2);
rect(x1, y1, ancho, alto);
ellipse(x1, y1, ancho, alto);
```

A continuación mostramos un ejemplo donde, utilizando las funciones antes mencionadas, podemos dibujar estas cuatro formas básicas.



Al analizar cómo se pintaron las formas básicas de manera automática, podemos entonces identificar dos elementos que conforman la instrucción:

```
line(4,0,8,4);
```

nombre de la función — parámetros

El nombre de la función es un identificador definido por Processing, donde le decimos cuál es la instrucción que utilizaremos. Processing cuenta con una lista de instrucciones, las cuales pueden ser consultadas en el manual de referencia en http://www.processing.org. Los parámetros, también llamados *argumentos*, brindan mayor flexibilidad a la función; en este caso, por medio de los parámetros podremos especificar las dimensiones de la línea (coordenada en x del punto 1, coordenada y del punto 1, coordenada x del punto 2 y coordenada y del punto 2). Nosotros no tendremos que especificar a qué corresponde cada uno de los valores, simplemente la instrucción identifica el orden de los argumentos para saber a qué corresponden.

Las formas básicas que definimos y dibujamos en papel son exactamente las mismas que deberemos utilizar en Processing para generar los programas.

El punto

El punto es la instrucción más simple. El punto lo utilizaremos para pintar un solo pixel en nuestra ventana de trabajo. De este modo, si quisiéramos pintar el pixel con las coordenadas 4, 5, la sintaxis sería la siguiente:

```
point(4,5);
```

La línea

Esta función permite dibujar una línea modificando las coordenadas (x, y) tanto del punto de origen como las coordenadas x, y del punto final. Esto nos permite controlar las dimensiones y la orientación de la línea. Si quisiéramos pintar la línea que fuera de la coordenada (3, 6) a la (10, 50), la sintaxis sería la siguiente:

```
line(3,6,10,50);
```

El rectángulo

Para poder dibujar un rectángulo, debemos utilizar la función *`rect()`*. A esta función podemos asociar cuatro parámetros, los cuales representan la coordenada superior izquierda donde nos interese ubicar el rectángulo y las dimensiones del rectángulo (ancho x alto). Si quisiéramos pintar un rectángulo en la coordenada (30, 30) con un ancho de 50 y una altura de 20, la sintaxis sería la siguiente:

```
rect(30,30,50,20);
```

De hecho, esta no es la única forma en que se puede representar el rectángulo en Processing, existen otras dos formas en las que podemos ubicar los parámetros en la instrucción, esto depende de cómo queremos dibujar dicho rectángulo.

Cuando dibujamos el rectángulo utilizando las coordenadas (x, y) como el punto superior izquierdo y damos el ancho y el alto (como lo hemos visto), estamos utilizando el mode CORNER.

```
rectMode(CORNER);
```

Cuando dibujamos el rectángulo utilizando las coordenadas (x, y) como punto central y definimos el ancho y el alto, estamos utilizando el modo CENTER.

```
rectMode(CENTER);
```

Cuando dibujamos el rectángulo utilizando dos puntos; las coordenadas (x, y) del primer punto y las coordenadas (x, y) del segundo punto, estamos utilizando el modo CORNERS.

```
rectMode(CORNERS);
```

La elipse

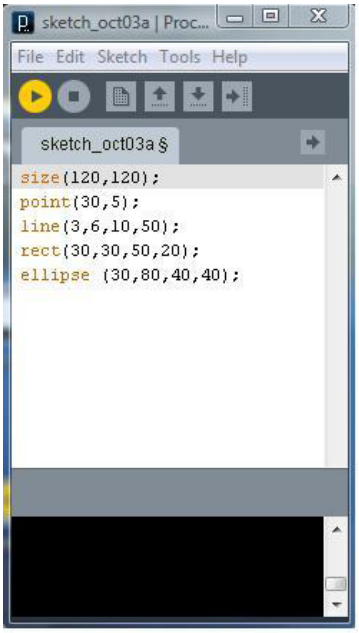
El funcionamiento de la instrucción elipse (*ellipse)* es muy similar a la del rectángulo *(rect)* antes vista. Para dibujar cualquier elipse, se sigue la técnica del Rectángulo Mínimo de Delimitación (MBR). Esta técnica se refiere a que cualquier elipse puede ser enmarcada siempre por un rectángulo mínimo. Teniendo esto en cuenta, debemos entonces verificar las coordenadas de este rectángulo. A la función *ellipse* podemos asociar cuatro parámetros, donde los dos primeros parámetros corresponden a las coordenadas (x, y) del punto central de la elipse y los dos parámetros restantes corresponden al ancho y al alto.

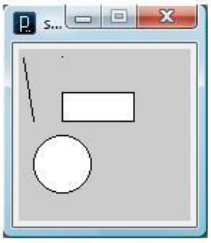
```
ellipse(30,80,40,40);
```

Esta forma de representación es el modo por default CENTER. Si quisiéramos cambiar de modo de dibujo, al igual que en la función *rect*, entonces tendríamos los siguientes modos de representación:

```
ellipseMode(CENTER);
ellipseMode(CORNER);
ellipseMode(CORNERS);
```

Este conjunto de instrucciones las podemos plasmar directamente en Processing para ver el resultado.





El punto (*point*), la línea *(line)*, la elipse *(ellipse)* y el rectángulo *(rect)* no son las únicas formas utilizadas por Processing, también existen las funciones de triángulo (triangle), cuadrilátero *(quad)* el arco *(arc)*. A continuación, se muestra su funcionamiento.

El triángulo

El triángulo se realiza conectando tres puntos. Los primeros dos argumentos se refieren a las coordenadas del primer punto, los dos argumentos siguientes se refieren a las coordenadas del segundo punto, para terminar con los últimos dos argumentos que definen las coordenadas del tercer punto.

```
triangle(0,90,45,45,70,70);
```

El cuadrilátero

El cuadrilátero es similar a la función *rect()*, solo que ahora debemos de dar un total de cuatro coordenadas diferentes, lo que provocará que los ángulos no necesariamente sean de 45 grados, esto quiere decir que el número de parámetros será de 8.

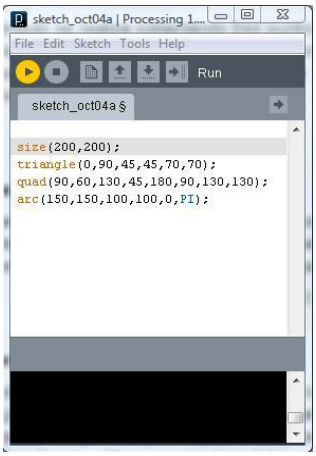
```
quad(90,60,130,45,180,90,150,85);
```

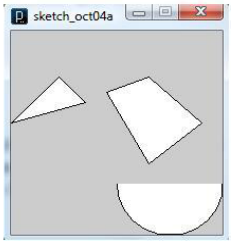
Los arcos

Para dibujar los arcos, se debe de tomar en cuenta la forma de trabajar de la función *ellipse()*. La función *arc()* utilizará seis parámetros. Los cuatro primeros son los mismos que para poder dibujar cualquier elipse, es decir, coordenadas del punto central x, y, el ancho y el alto. Para los dos últimos parámetros, debemos de escribir el segmento inicial y final que queremos dibujar. Estos segmentos están definidos en radianes (0, PI,TWO_PI, etc.).

```
arc(250,250,100,100,0,PI);
```

Existen tres formas diferentes de dibujar el arco, que están definidas por un séptimo parámetro. Estas tres formas son `OPEN`, la cual dejará completamente abierto el arco y el modo por default, el modo `PIE`, que completará el arco tomando en cuenta el centro del arco y tomará la forma de *pie*, mientras que el tercer modo es `CHORD`, el cual genera un línea que cierra el semicírculo desde el punto de origen hasta el punto final. A continuación, se muestra la manera como se ven estas nuevas formas geométricas directamente `en Processing.`





EL SIZE

Otra función muy importante es `size()`, que permite definir el tamaño de la ventana donde vamos a desplegar todos los elementos visuales generados por el programa. Esta función generalmente se ubica en la parte superior de los programas. Los parámetros utilizados por la función son el número de pixeles correspondientes al ancho y el alto de la ventana.

LA ESCALA DE GRISES

Digitalmente, el color negro corresponde al valor 0 y el color blanco al valor 255. Esto quiere decir que existen, entre estos dos valores, 254 diferentes colores que podríamos representar. Dichos valores corresponden entonces a una escala de grises. Con esto en mente, debemos inicialmente comprender tres instrucciones adicionales, mediante las cuales podremos aplicar la escala de niveles de gris. Estas instrucciones son `background()`, `fill()` y `stroke()`.

El background

La instrucción `background()` permite definir la escala de grises deseada que utilizaremos para el fondo de pantalla donde se desplegarán cada una de las formas geométricas utilizadas.

```
background(10);
```

El stroke

La función `stroke()` permite definir la escala de grises deseada para los contornos de las formas geométricas.

```
stroke(255);
```

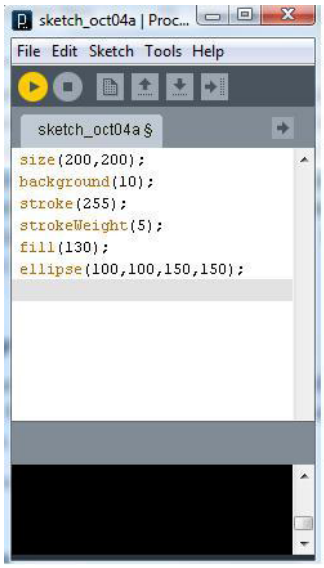
La función `stroke()` permite trabajar con los contornos de las formas, existe la función `noStroke()`, que desaparece totalmente el contorno. También podemos utilizar la función `strokeWeight()` para definir el grosor del contorno deseado.

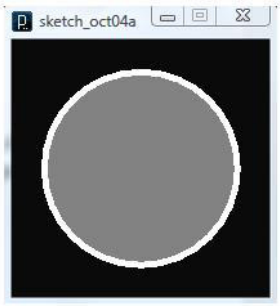
El fill

La instrucción `fill()` permite modificar el nivel de gris de la parte interna de las formas. Este es un relleno que se utiliza de la siguiente manera:

```
fill(130);
```

Asimismo, está la función `noFill()`, la cual eliminará por completo el relleno de las formas y permitirá visualizar únicamente el contorno. A continuación, se muestra un ejemplo del uso de estas instrucciones.

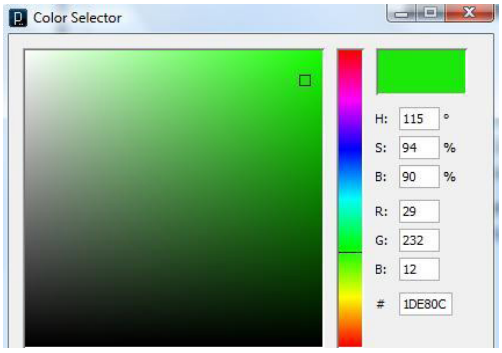




EL COLOR

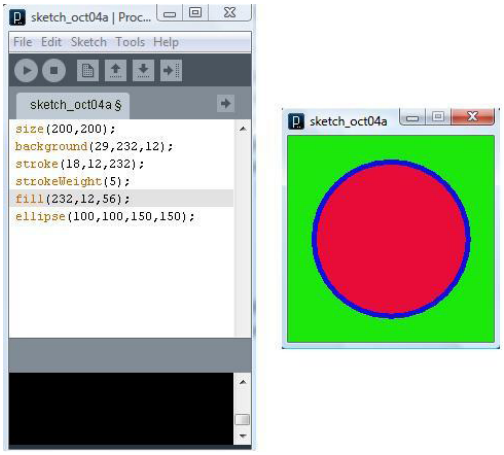
El color, en el mundo digital, es generalmente construido al mezclar tres colores primarios (rojo, verde, azul), en inglés RGB (Red, Green, Blue). Processing no es la excepción. Para generar algún color, tenemos que formar una combinación de estas tonalidades.

Processing cuenta con una herramienta llamada *color selector*, la cual puede ser desplegada a partir del menú *Tools*, como se muestra a continuación:



En *Color Selector* podemos observar tres diferentes formas de escoger el color deseado, utilizando el modo HSB (Hue, Saturation y Brightness), el RGB o, finalmente, utilizando la codificación Hexadecimal correspondiente al color deseado. Por default, las funciones se encuentran definidas con el RGB.

Para ingresar esta codificación del color podemos modificar el valor del nivel de gris, definido en las funciones `background()`, `fill()` y `stroke()`. De este modo, en lugar de definir un parámetro, podremos definir tres.



LA TRANSPARENCIA

Además de definir cualquier combinación de color, Processing permite definir el nivel de transparencia de cada uno de los colores. Este nivel va del 0 (color completamente sólido) al 255 (color completamente transparente). Este nivel de transparencia, muchas veces llamado nivel *alpha, se define como un cuarto parámetro dentro de las funciones* `background()`, `fill()`, y `stroke()`.

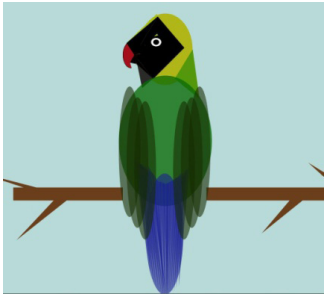
Como recordaremos en los capítulos de este material, al visitar el manual de referencia de la página http://processing.org se puede obtener una gran ayuda para terminar de comprender los conceptos e instrucciones vistos en cada capítulo.

Actividad Integradora

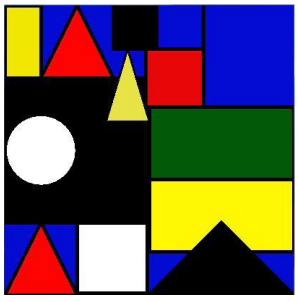
Realizar un dibujo libre utilizando las primitivas básicas y el color, donde se plasme algún área común de la universidad (biblioteca, canchas deportivas, comedor, áreas al aire libre, etc.)

Experiencias previas

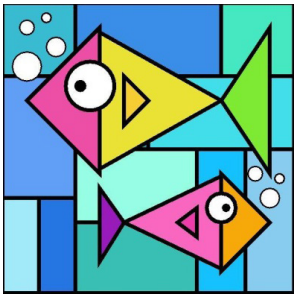
Dibujo libre usando figuras básicas



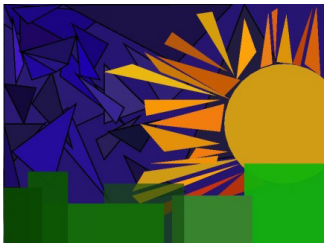
Autor: Juan Carlos Cabrera Romero
Ciencias de la Comunicación
Trimestre 13 Invierno



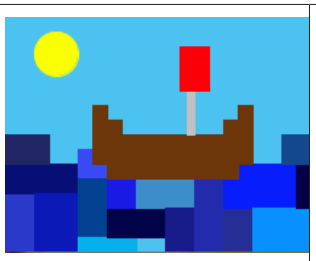
Autor: Camila Mata Lara
Ciencias de la Comunicación
Trimestre 13 Invierno



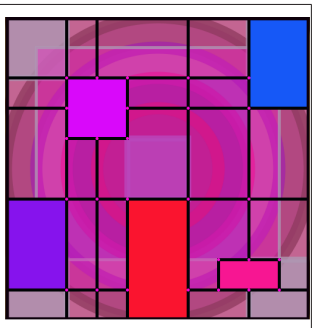
Autor: Rebeca Ortiz Santibáñez
Ciencias de la Comunicación
Trimestre 13 Invierno



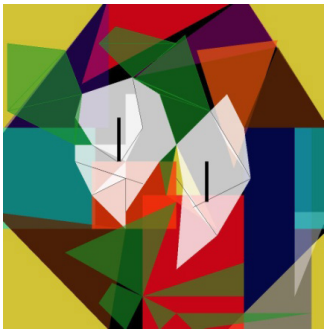
Autor: Adriana Fabián Chávez
Ciencias de la Comunicación
Trimestre 13 Invierno



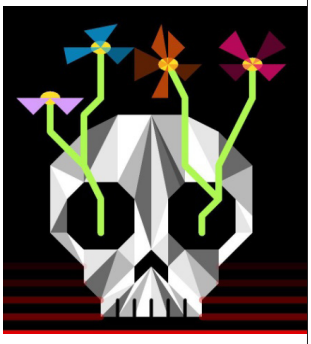
Autor: Dalia
Castañeda Núñez
Ciencias de la Comunicación
Trimestre 14 Otoño



Autor: Juan Luis Petrearce
Ciencias de la Comunicación
Trimestre 14 Otoño



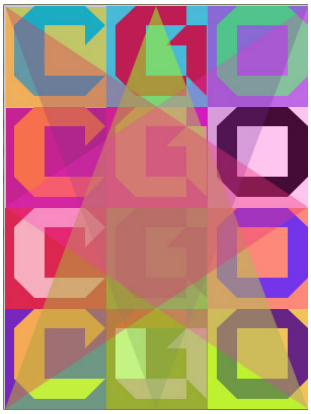
Autor: Diana Morán
Ciencias de la Comunicación
Trimestre 14 Otoño



Autor: María José
Ros Gómez
Ciencias de la Comunicación
Trimestre 14 Otoño

Autor: Román Morales
Enríquez
Ciencias de la Comunicación
Trimestre 14 Otoño

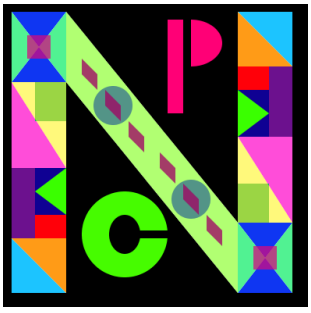
Experiencias previas
Dibujar iniciales de nombre utilizando formas básicas



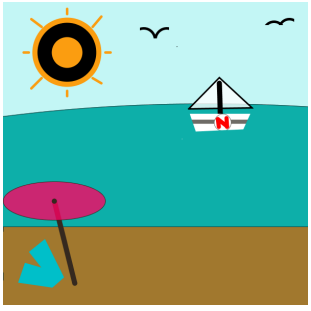
Autor: Carlos Gerardo Orozco
Ciencias de la Comunicación
Trimestre 14 Otoño

Autor: Abraham Cortés Gómez
Ciencias de la Comunicación
Trimestre 14 Otoño

Autor: Teresa Escamilla Paredes
Ciencias de la Comunicación
Trimestre 14 Otoño



Autor: Natalia Castañeda Páez
Ciencias de la Comunicación
Trimestre 14 Otoño



Autor: Oscar Nieto Villegas
Ciencias de la Comunicación
Trimestre 14 Otoño

Autor: María Alexia Cervantes Díaz
Ciencias de la Comunicación
Trimestre 14 Otoño

Capítulo 3. Interacción

"Los ordenadores son inútiles. Solo pueden darte respuestas"
Pablo Picasso

Objetivo

- El alumno sabrá cómo organizar su programa a partir de las funciones `setup()` y `draw()`.
- El alumno conocerá dos principales funciones, `mousePressed()` y `keyPressed()`, para dar al usuario la posibilidad de interactuar con el programa.

Actividades de aprendizaje

- El alumno identificará dos bloques de cualquier programa con los cuales (1) dará instrucciones de configuración e inicialización de un programa y (2) dará instrucciones de dibujo.
- El alumno utilizará el mouse y el teclado para interactuar con el programa a partir de las funciones `mousePressed()` y `keyPressed()`.
- El alumno construirá programas en los que pueda visualizar el sentido de las funciones `setup()` y `draw()`.

Función `setup()` y `draw()`

Processing permite trabajar en dos modos de programación: uno estático y otro dinámico. En el primer modo, se escriben las instrucciones en orden descendente pero sin una organización externa, mientras que en el segundo modo de programación el código se organiza en estructuras.

Hasta el momento, las instrucciones que se han escrito en Processing no han sido separadas de acuerdo con el modo de programación: (1) instrucciones de configuración e inicialización del programa y (2) instrucciones de dibujo. En este capítulo, el alumno aprenderá a separar las instrucciones de acuerdo con su funcionalidad, así como a interactuar con el programa a partir del uso del mouse y el teclado.

Cuando se realizan dibujos utilizando Processing, se piensa inmediatamente en la posibilidad de poder interactuar con dichos dibujos. Asimismo, es importante ir más allá de los dibujos estáticos. En este caso, tomando como conocimiento previo la manera en la que se dan instrucciones (paso a paso), podemos decir que todos los programas tienen un flujo (serie de instrucciones), un inicio y un fin. Por ejemplo:

> Paso 1. Doy las condiciones de inicio
> Paso 2. Sigo las instrucciones, una y otra vez, hasta que el programa termina

En un ejemplo más concreto, podemos tener no solo condiciones de inicio, sino también instrucciones que se repiten muchas veces hasta que se cumple determinado objetivo. Por ejemplo, si vamos a participar en una carrera de 10 kilómetros, algunas de las instrucciones que seguiríamos serían:

> Paso 1. Me pongo tenis y hago estiramientos
> Paso 2. Empiezo a correr dando un paso tras otro. Repito esto lo más rápido posible.
> Paso 3. Al llegar al décimo kilómetro me detengo.

En el ejemplo anterior, la condición de inicio está representada con el paso 1. Es decir que no podemos empezar a correr si no hay algunas condiciones previas. El paso 2 corresponde a un ciclo en el cual se repiten las condiciones una y otra vez hasta que sucede algo. Ese suceso, que detiene el paso 2, es llegar a la meta. Aquí, el objetivo final es correr 10 kilómetros, por lo que se repetirá el paso 2 hasta que se haya logrado dicho objetivo.

En un programa computacional, también debemos de preparar el escenario y realizar algunas instrucciones que *inicializan* el proceso antes de dar paso a la serie de instrucciones, mediante las cuales alcanzaremos un determinado objetivo.

En Processing, la serie de instrucciones que inicializan están dentro de una función llamada `setup()`. Como su nombre lo dice, `setup()` permite configurar diversas opciones de acuerdo con las necesidades del usuario. Por su parte, la función que agrupará una serie de instrucciones de dibujo hasta alcanzar un objetivo se llama `draw()`. En esta instrucción se agrupan las instrucciones que se realizarán muchas veces. En cambio, en `setup()` las instrucciones solo se realizan la primera vez.

```
void setup()
{
// instrucciones de inicialización
}

void draw()
{
// instrucciones de dibujo
}
```

← Algunas instrucciones de setup pueden ser: `size()` (tamaño de la pantalla), `background()`, etc.
Estas instrucciones solo *se ejecutan una sola vez.*

← En este bloque del programa van las instrucciones que *se realizan muchas veces* hasta que el programa llega a su fin.

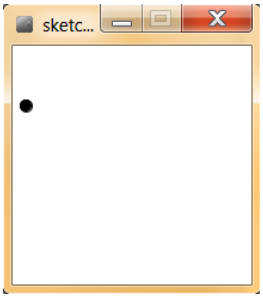
A continuación, mostramos un ejemplo en el que se utiliza `void setup()` y `void draw()`, y se constata con un ejemplo donde el código no está dentro de estos bloques. El objetivo del ejemplo es dibujar una elipse.

(a) Código sin bloques

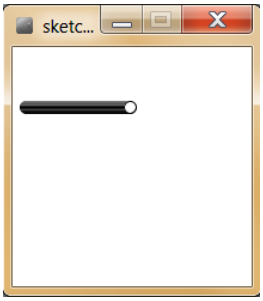
```
int a;
a = 10;
size(200,200);
background(255);
fill(0);
a = (a+1) % 200;
ellipse(a,50,10,10);
```

(b) Código dentro de bloques (funciones)

```
int a;
void setup()
{
    a = 10;
    size(200,200);
    background(255);
}
void draw()
{
    a = (a+1) % 200;
    ellipse(a,50,10,10);
}
```



(a)



(b)

En el caso (a), la elipse solo se dibuja una vez; en cambio, en el caso (b), se dibuja tantas veces mientras no se presione el botón de stop. Es decir que el `draw()` funciona como un ciclo, las instrucciones se repiten una y otra vez.

VARIABLES DEL SISTEMA MOUSEX Y MOUSEY, PMOUSEX Y PMOUSEY

En un lenguaje de programación, una ***variable*** (ver capítulo 4) es un espacio de almacenamiento para un resultado. Existen dos tipos de variables:

- las *variables definidas por el usuario*: el usuario las crea para ir guardando resultados y
- las *variables del sistema*: estas son predefinidas por el lenguaje de programación y están ya asociadas a una determinada funcionalidad.

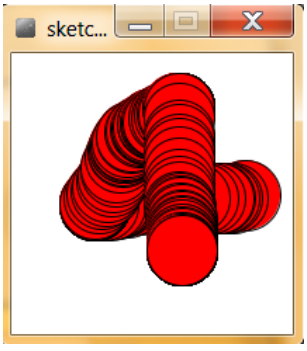
En este apartado, presentamos cuatro variables del sistema (mouseX, mouseY, pmouseX, pmouseY) que permiten una interacción con el programa realizado. A pesar de que hasta estos momentos no se ha impartido la clase de variables, este tema le ayuda al alumno a comprender la dimensión de la variable con ejemplos prácticos.

La interacción es mediada a partir de los dispositivos externos con los que se cuenta. El teclado o el mouse son dispositivos externos que nos permiten interactuar con los programas. Generalmente, cuando se utiliza un mouse, podemos moverlo incluso mientras se corre un programa. Sin embargo, el movimiento del mouse es captado únicamente si le decimos al programa que queremos captar la posición del mouse. ¿Esto de qué nos sirve? El poder captar la posición del mouse, a partir de su coordenada en X y en Y, permite interactuar con el programa para indicar una posición, dibujar de acuerdo con la posición del mouse, etc.

Para detectar la posición del mouse, existen dos variables del sistema mouseX (posición horizontal) y mouseY (posición vertical). Debido a que estas variables son definidas por el sistema, no es necesario que el usuario las defina ni les atribuya algún tipo. En el siguiente ejemplo, se juega con el uso de `background` en `void draw()` y en `void setup()` con el fin de observar, claramente, el uso del mouse.

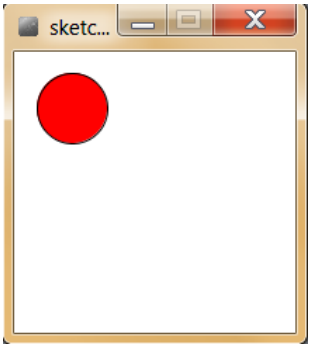
(a) Background en **`void setup()`**

```
int x,y =100;
void setup()
{
    background(255);
    size(200,200);
}
void draw()
{
    fill(255,0,0);
    x=mouseX-25;
    y=mouseY-25;
    ellipse(x,y,50,50);
}
```



(b) Background en **`void draw()`**

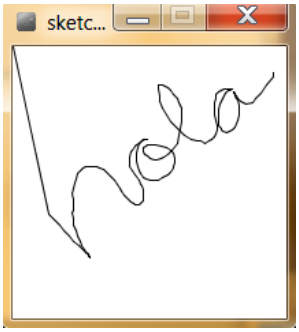
```
int x,y =100;
void setup()
{
    size(200,200);
}
void draw()
{
    background(255);
    fill(255,0,0);
    x=mouseX-25;
    y=mouseY-25;
    ellipse(x,y,50,50);
}
```



En el caso (a) en el que el `background` está definido en el `setup()`, se puede ver el recorrido que hace la elipse debido al movimiento del mouse. En este caso, la coordenada (x,y) toma los valores del `mouseX` y `mouseY`. En el caso de la elipse del segundo ejemplo, se ve una sola elipse debido a que el `background` sirve como una especie de goma que limpia la pantalla en cada ciclo del `draw`.

Otras variables definidas por el sistema corresponden a `pmouseX` y `pmouseY`. Estas toman el valor anterior de `mouseX` y `mouseY` para mostrar una continuidad en el trazo. En el siguiente ejemplo, usando `pmouseX` y `pmouseY`, se simula un lápiz.

```
void setup()
{
    size(200, 200);
    background(255);
}
 void draw()
{
    stroke(0);
    line(pmouseX, pmouseY, mouseX, mouseY);
}
```



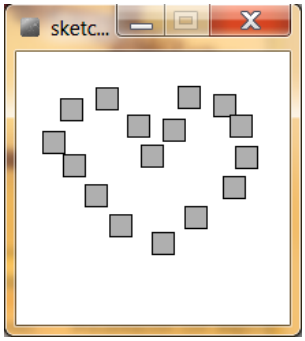
No solamente el movimiento del mouse provoca una interacción con el programa, sino que también es posible usar el clic del mouse y el teclado.

Interacción con el mouse

Es muy importante poder interactuar con el mouse y, sobre todo, tener control sobre este a partir del clic. El clic es considerado un *evento que genera instrucciones específicas definidas por el programador. Un evento debe de ser manejado en un bloque de código independiente de* `void setup()` y de `void draw()`. Por lo tanto, si el usuario interactúa con el clic del mouse, este clic debe de ser registrado como un evento que genera que algunas instrucciones sean ejecutadas. Para esto, se utiliza el bloque `void mousePressed()`. Cada vez que el usuario dé un clic, entonces, se ejecutarán las instrucciones contenidas dentro del bloque.

En el siguiente ejemplo se desea que aparezcan pequeños cuadros en la pantalla de acuerdo con la posición del mouse. Sin embargo, al estar el código localizado en el bloque de `void mousePressed()`, estos cuadros solo aparecerán cuando se haga un clic.

```
void setup()
{
    size(200,200); background(255);
}
void draw()
{
}
void mousePressed()
{
    stroke(0);
    fill(175);
    rectMode(CENTER);
    rect(mouseX,mouseY,16,16);
}
```



Es posible formar figuras y poner los pequeños cuadros en el lugar que el usuario desea a partir de posicionarse dentro de la pantalla y dar clic cuando la posición sea la adecuada.

INTERACCIÓN CON EL TECLADO

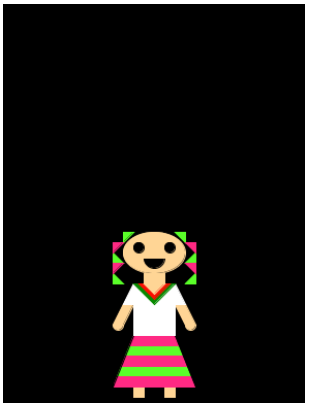
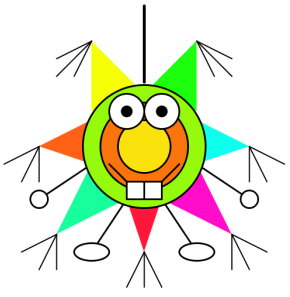
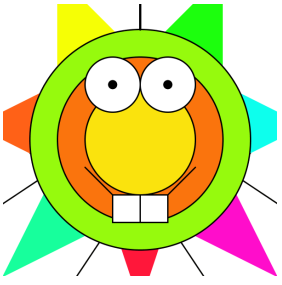
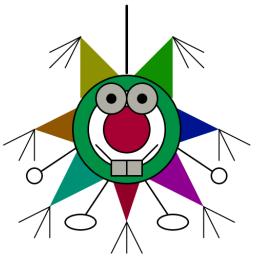
No solo es posible detectar si se ha ejecutado un clic proveniente del mouse o la posición del mouse, además es posible detectar si una tecla ha sido presionada. El programador puede definir ciertas funcionalidades de acuerdo con la tecla oprimida. Sin embargo, por el momento, el teclado puede funcionar independientemente de la tecla oprimida. La función asociada al teclado es `keyPressed()`, que al igual que `mousePressed()`, genera un evento el cual hace que durante el programa se vaya al bloque de `keyPressed()` para ejecutar las instrucciones que contiene. Generalmente, dentro del curso se asocia a poder reinicializar valores o a borrar la pantalla (uso de background). En las siguientes experiencias previas, se conjuntan varios conocimientos aprendidos durante el curso: color, separación del código en bloques, uso de `keyPressed()`, uso de `mousePressed()`.

Actividad integradora

Realizar un dibujo libre utilizando los conceptos que se estudiaron en el presente capítulo: formas, colores, movimiento con mouse, `keyPressed()` y `mousePressed()`. El dibujo libre debe de contar con al menos tres elementos que tengan interactividad.

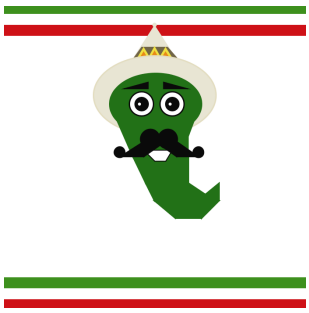
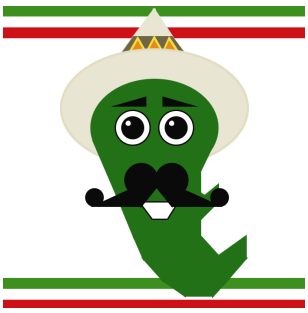
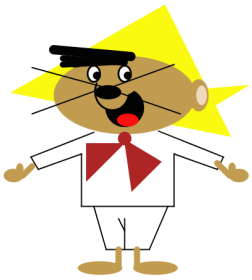
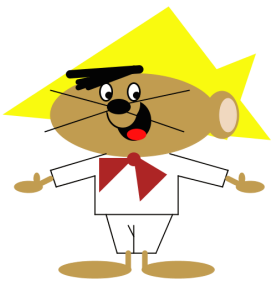
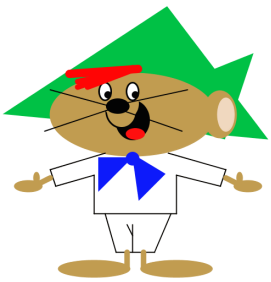
Experiencias previas

Mascota mexicana que, al oprimir una tecla, cambie de color, y si se da un clic, cambie de tamaño.

Autor: Natalia Castañeda Páez Ciencias de la Comunicación Trimestre 14 Otoño 	Usando `mousePressed()`: 	Usando `keyPressed()`: 
Autor: Rubén Darío Martínez Ciencias de la Comunicación Trimestre 14 Otoño 	Usando `mousePressed()`: 	Usando `keyPressed()`: 

Experiencias previas

Mascota mexicana que, al oprimir una tecla, cambie de color, y si se da un clic, cambie de tamaño.

Autor: Ilse López González Ciencias de la Comunicación Trimestre 14 Otoño	Usando `mousePressed()`:	Usando `keyPressed()`:
		
Autor: Xóchitl Alelí Bastida Lara Ciencias de la Comunicación Trimestre 14 Otoño	Usando mousePressed():	Usando keyPressed():
		

En este capítulo se presentó el uso de bloques dentro de un programa. Dichos bloques permiten controlar (1) qué forma parte de la inicialización del programa usando `setup()`, el cual solo se realiza una vez y (2) qué forma parte de lo que se va a estar dibujando usando `draw()`, el cual se realiza varias veces hasta que el programa es detenido. De igual forma, se presentaron las variables del sistema `mouseX`, `mouseY`, `pmouseX` y `pmouseY` que captan el movimiento del mouse. Otra manera de interactuar con el programa es por medio del clic del mouse usando `mousePressed()`, y mediante el teclado usando `keyPressed()`.

Capítulo 4. Variables

"Controlar la complejidad es la esencia de la programación"
Brian Kernigan

Objetivo

- Entender el concepto de variable.
- Conocer los tipos de datos utilizados en computación y aplicarlos en programas simples.

Actividades de aprendizaje

- Conocer la necesidad de utilizar variables.
- Identificar el tipo de variables a utilizar.
- Realizar ejemplos aplicando las variables.
- Utilizar las variables del sistema.
- Utilizar el `random` para generar formas aleatorias.

Las variables

La variable es un elemento frecuentemente utilizado en la computación y, en general, en todos los lenguajes de programación. El concepto básico de la variable se relaciona con la capacidad de poder almacenar un valor dentro del programa. Este valor podrá utilizarse y modificarse en la ejecución del programa.

Una variable está formada por un espacio físico en la memoria de la computadora, su valor, y un nombre simbólico, comúnmente llamado *identificador*, que está asociado a dicho espacio. El valor y el identificador de la variable permiten que el identificador sea usado independientemente de la información exacta que representa. El identificador, en el programa, puede estar ligado a un valor durante el tiempo de ejecución y el valor de la variable puede por lo tanto cambiar durante el curso de la ejecución del programa.

Un ejemplo que todos conocemos en la vida cotidiana de la utilización de variables es la tabla de posiciones que tienen los equipos de futbol. Cada equipo tiene un identificador: América, Tijuana, Cruz Azul, Chivas, etc. En la tabla de posiciones, para cada equipo tenemos el número de puntos (su valor) según la jornada que se jugó. Este valor va cambiando jornada tras jornada, pero el identificador (nombre del equipo) se mantiene. Esto es exactamente lo que sucede en un programa de cómputo: el identificador se mantiene pero su valor asociado cambia constantemente con la ejecución del mismo.

El nombre del identificador será proporcionado por el programador de manera libre y deberá ser lo suficientemente descriptivo para poder entender qué es lo que se almacenará. La única restricción es la utilización de alguna palabra reservada por Processing.

El valor que se almacenará en el identificador tiene que ser definido, en otras palabras debemos de seleccionar qué tipo de dato a utilizar. A continuación explicaremos los diversos tipos de datos utilizados por Processing.

TIPOS DE DATOS

El tipo de dato es un atributo que indica a la computadora (y/o al programador) qué datos se van a procesar. En el caso de las variables, indica qué tipo de información podemos almacenar en el identificador y qué operaciones podemos realizar. Los tipos de datos comunes son: enteros, flotantes (decimales), caracteres, cadenas de caracteres, etc.

- int
  El tipo de dato int representa un conjunto de enteros de 32 bits, así como las operaciones que se pueden realizar con los enteros, como la suma, la resta y la multiplicación. Su rango va del -2.147.483.648 al 2.147.483.647.
- float
  El tipo de dato float se refiere a la utilización de números con punto decimal. El rango de valores va del -3.40282347E+38 al 3.40282347E+38 y está almacenado en 32 bits.
- char
  El tipo de dato char permite almacenar cualquier tipo de carácter (letras o símbolos) en el formato Unicode.
- boolean

El tipo de dato boolean permite únicamente el almacenamiento de dos valores: TRUE y FALSE. Este tipo de dato es muy útil para el determinar el flujo de los programas.

- color
  El tipo de dato color permite el almacenamiento de cualquier color codificado en números hexadecimales.

Declaración e inicialización

Las variables que se van a utilizar deben de ser declaradas en todo programa. En otras palabras, necesitamos avisarle al programa que vamos a utilizarlas y que, por lo tanto, hay que crearlas. La declaración de las variables se realiza de la siguiente manera:

```
int variable1;       // para una variable entera
float diametro;    // para un variable flotante
char letra;            // para una variable carácter
boolean valor;       // para una variable booleana
color arcoiris;  // para una variable color
```

Si las variables van a ser utilizadas en todo el programa, se deberán declarar al inicio del programa, antes de la función `setup()`, y se les nombrará variables *globales*. Si las variables se utilizan únicamente en alguna función, entonces deberán declararse al inicio de la función deseada y se llamarán variables *locales*. Estas variables únicamente existirán cuando la función esté siendo utilizada.

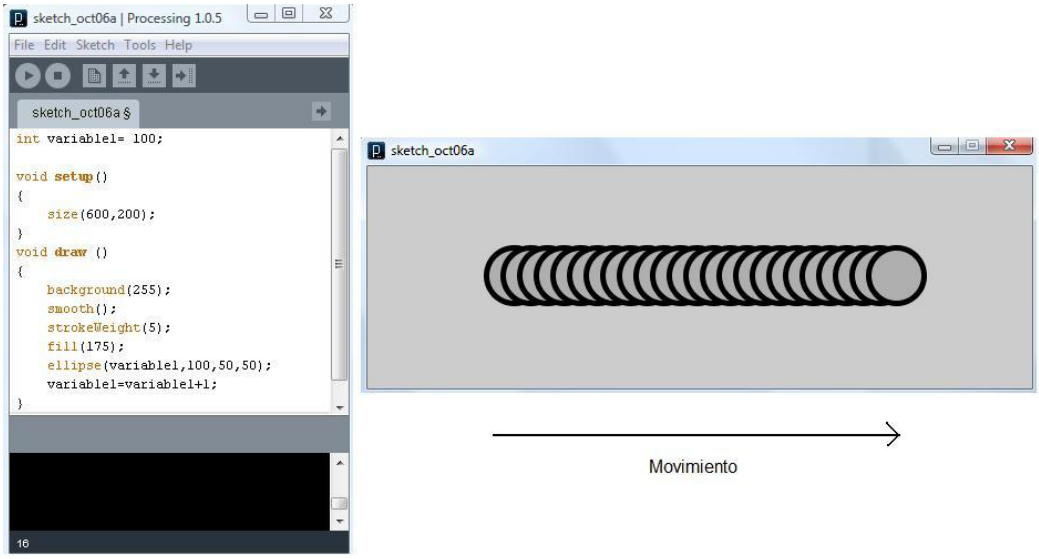
Las variables deben de tomar su valor mediante una llamada *asignación* del valor. El símbolo para realizar esta asignación es el =. En computación, este símbolo no significa *igual que* y no estamos haciendo ningún tipo de comparación entre dos valores. La asignación se realiza modificando el valor de la variable por el valor o por el resultado de la parte derecha del símbolo =. De este modo, algunas asignaciones comunes son las siguientes:

```
int count =0;              //Asignamos el valor 0 a la variable count
char letter = 'a';         //Asignamos el valor 'a' a la variable letter
float d= 132.32;          //Asignamos el valor 132.32 a la variable d
float x = 4.0;            //Asignamos el valor 4.0 a la variable x
float y = x + 5.2;        //Asignamos el valor 9.2 a la variable y
float z = x*y + 15.0;     //Asignamos el valor 51.8 a la variable z
```

Es importante hacer notar que, por ejemplo, en algunos casos los valores de las variables se adquieren mediante el resultado de una operación realizada en la parte derecha del símbolo =. Eso pasa, por ejemplo, en la asignación del valor de la variable z.

Una vez asignado un valor en la variable, entonces esta se encuentra lista para ser utilizada en cualquier sección de nuestro programa. La idea de la variable es generar un dinamismo en el programa y que no sea completamente estático. Para lograrlo, la variable tendrá que ser utilizada en el programa, sustituyendo normalmente algún valor estático utilizado.

A continuación, se muestra un ejemplo del uso de una variable.



En este ejemplo podemos identificar tres características importantes. La primera es la declaración de la variable, la cual se encuentra en la parte superior del programaLa segunda es el uso de la variable, donde se sustituye un valor estático en la función elipse, por la variable, en este caso, la posición x de la elipse. La tercera es el uso de la variable, que en este caso es el incremento de la variable misma en 1. Estas características permiten que la elipse tenga un desplazamiento constante hacia la derecha.

Así, se deduce que esta variable puede ser muy útil para generar cambios en el comportamiento del programa. Podríamos utilizar la variable para

sustituirla por cualquier valor estático (argumento) que se encuentre dentro de la función `draw()`, lo que generaría cambios en el comportamiento del programa.

Se pueden generar tantas variables como sean necesarias, y actualizarlas según los requerimientos del programa.

Ejercicio propuesto

Utilizando el programa anterior, ¿cómo lo modificarías para realizar las siguientes acciones?

- Que la elipse se desplace de derecha a izquierda
- Que la elipse crezca de tamaño de manera uniforme
- Que la elipse cambie de color
- Que el fondo de la pantalla cambie de color
- Que la elipse crezca y, a su vez, se desplace hacia abajo

LAS VARIABLES PREDEFINIDAS (DEL SISTEMA)

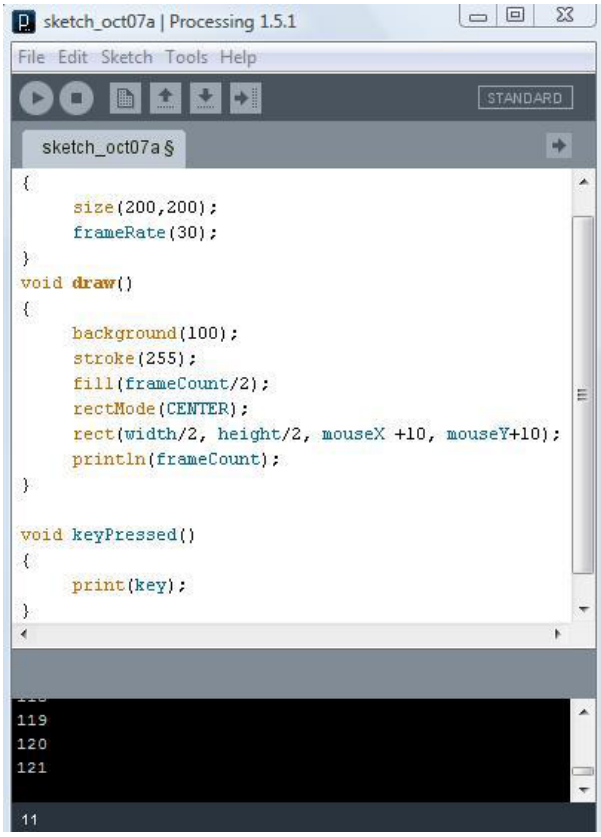
Existen un conjunto de variables que están predefinidas por Processing. Estas variables tienen la ventaja de que no necesitan estar declaradas, simplemente se utilizan directamente por los programas. Todas estas variables predefinidas son de mucha utilidad para la ejecución y el seguimiento de los programas. Las más importantes son:

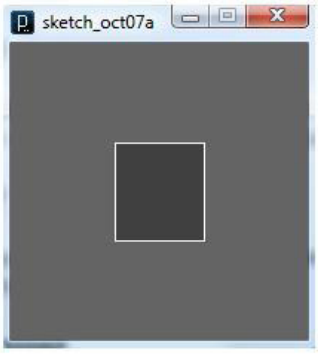
- width
  Esta variable almacena el ancho de la ventana de trabajo y se inicializa utilizando el primer parámetro de la función `size()`.
- height
  Esta variable almacena el alto de la ventana de trabajo y se inicializa por el segundo parámetro definido en la función `size()`.
- frameRate
  En la variable `frameRate` se guarda la velocidad con la que se está ejecutando el programa o sketch. Se puede definir la velocidad de ejecución mediante la función `frameRate();`
- frameCount
  La variable `frameCount` contiene el número de *frames* que el programa ha ejecutado desde su inicio. Esta variable se inicializa con el valor

de cero y al entrar a la función `draw()` obtiene el valor de 1, y así sigue aumentando constantemente.

- displayHeight
  Esta variable almacena el alto de la pantalla completa a ser desplegada. Es utilizada para crear una ventana, independientemente del dispositivo de ejecución.
- displayWidth
  Esta variable almacena el ancho de la pantalla completa a ser desplegada. Funciona de igual forma que `displayHeight` pero para el ancho.
- key
  La variable del sistema `key` contiene el valor reciente de la tecla oprimida desde el teclado.
- keyCode
  Esta variable es utilizada para detectar teclas especiales como las flechas UP, DOWN, LEFT o RIGHT, o teclas como ALT, CONTROL, o SHIFT.
- keyPressed
  Esta variable del sistema es de tipo booleano, por lo que tendrá el valor de TRUE si alguna tecla es oprimida y FALSE para el caso contrario.
- mouseX
  Esta variable siempre conserva la coordenada horizontal del mouse. Esta información es únicamente utilizada cuando el mouse está dentro de la ventana de trabajo. Inicialmente el valor está definido como 0.
- mouseY
  La variable `mouseY` funciona exactamente de la misma forma que `mouseX`, solo que almacena la coordenada y la posición actual del mouse.
- pmouseX
  La variable `pmouseX` contiene la posición horizontal del mouse en el *frame* anterior al actual. En otras palabras, almacena la posición anterior del mouse.
- pmouseY
  Al igual que la variable `pmouseX`, esta variable almacena la posición vertical del mouse en el *frame* anterior al actual.
- mousePressed
  Esta variable booleana se encarga de almacenar si el botón del mouse está actualmente oprimido o no. El valor de TRUE es cuando el botón está siendo oprimido y el valor de FALSE cuando se libera el botón.
- mouseButton
  La variable `mouseButton` detecta qué botón del mouse fue seleccionado. Adquiere entonces los valores de LEFT, RIGHT o CENTER, dependiendo del botón seleccionado. Inicialmente tiene el valor de 0.

A continuación, se muestra un ejemplo que contiene el uso de algunas de estas variables.





En este ejemplo, un conjunto de variables del sistema fueron utilizadas. Se empezó con `frameRate`, que es la velocidad con que se ejecutará el programa; después siguió el `frameCount`, que es el número de veces que se ha ejecutado el `draw()`, el `width` y el `height` para determinar el tamaño de la pantalla, `mouseX` y `mouseY` para el tamaño cambiante del rectángulo y la variable `key` para poder imprimir la tecla oprimida por el usuario.

EL RANDOM

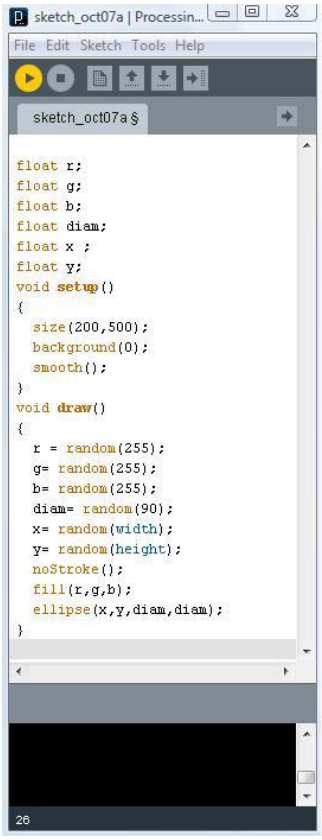
La función `random` se refiere al proceso de aleatoriedad. Este término se asocia a todo proceso cuyo resultado no es previsible más que por azar. El resultado de todo suceso aleatorio no puede determinarse en ningún caso antes de que este se produzca. En computación, la función `random` es capaz de generar un número flotante aleatorio dado un rango de valores proporcionado.

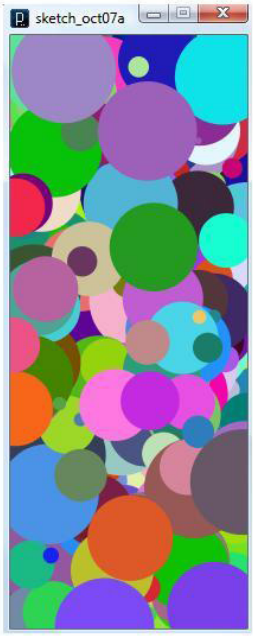
La función `random` se utiliza de la siguiente manera:

```
float w;
w=random(0,10);
```

donde la variable w es definida como flotante y después a esa variable le asignamos el valor random.

A continuación, se muestra un ejemplo de uso de esta función, donde todos los elementos se generan de manera aleatoria (color, tamaño, ubicación).



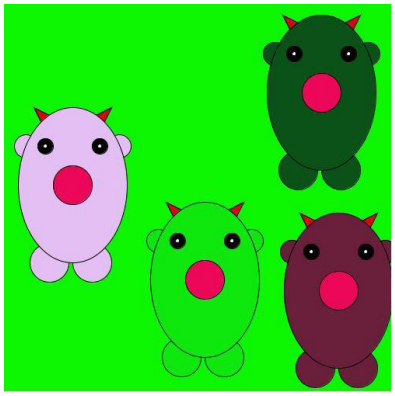


Actividad integradora

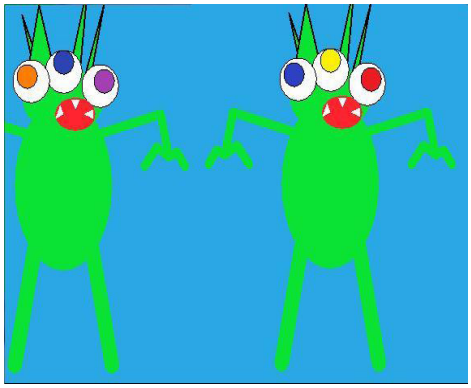
Utilizando los elementos vistos en el capítulo (variables, variables del sistema), genera un protector de pantalla que genere elementos visuales aleatorios, dado un determinado tiempo y a una determinada velocidad.

Experiencias previas

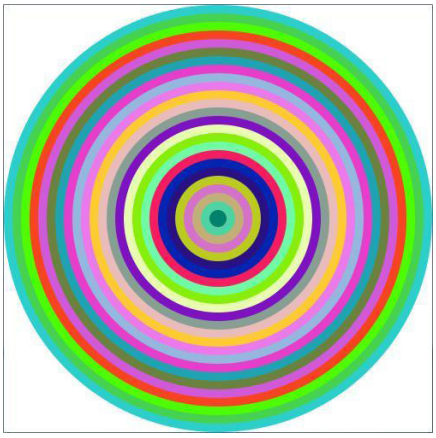
Generación aleatoria de colores y movimiento



Autor: Emilio Trasviña Díaz
Ciencias de la Comunicación
Trimestre 13 Invierno

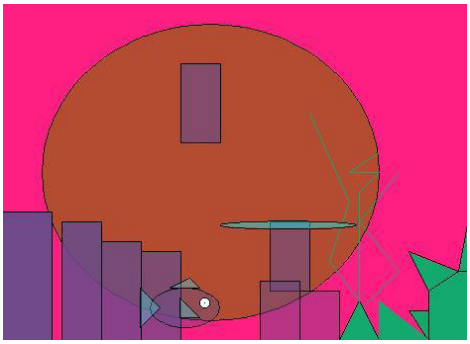


Autor: Uriel Espinoza Alberdín
Ciencias de la Comunicación
Trimestre 13 Invierno

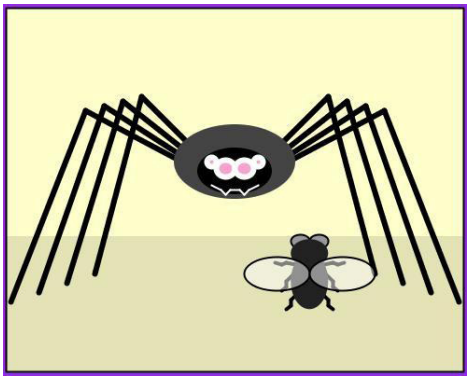


Autor: José Luis Alvarado Negrete
Lic. en Diseño
Trimestre 10 Invierno

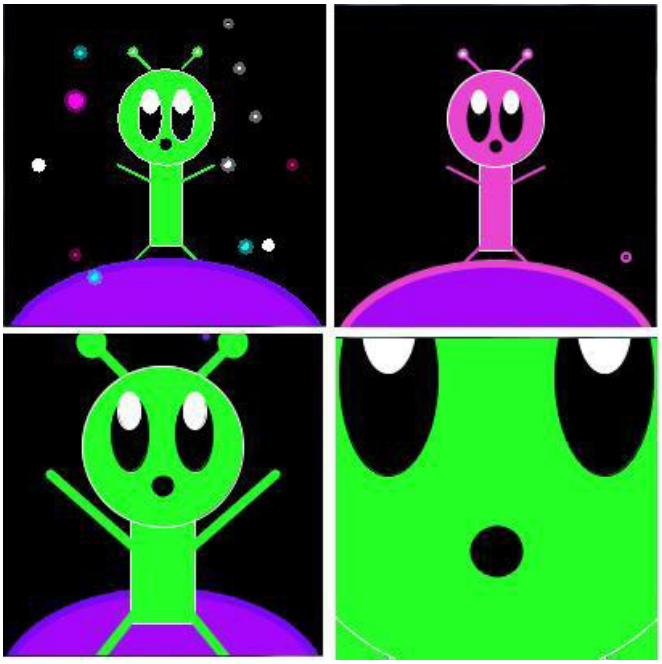
Autor: Itzel Díaz Villagrán
Ciencias de la comunicación
12 Otoño



Autor: Leslie López Brito
Ciencias de la comunicación
12 Otoño



Autor: Manuel Rodríguez González
Ciencias de la comunicación
12 Otoño



Autor: Laura Álvarez Ortega
Ciencias de la Comunicación
12 Otoño

Capítulo 5. Condicionales

"Lo mejor de los booleanos es que si te equivocas,
estás a un solo bit de la solución correcta"
Anónimo

Objetivo

- Entender el concepto de condicionales.
- Identificar los diversos casos de utilización de las condicionales.
- Entender la generación de botones mediante las condicionales.

Actividades de aprendizaje

- Identificar la necesidad de utilizar las condicionales.
- Estudiar los operadores lógicos utilizados en las condicionales.
- Realizar ejemplos aplicando las condicionales.
- Implementar botones utilizando condicionales.

Las condicionales

Después de haber visto el concepto de variables y sus múltiples usos, es momento de estudiar uno de los pilares de la programación estructurada: las condicionales.

Una condicional, en la programación, es una sentencia o grupo de sentencias que puede ejecutarse, o no, en función del valor de una condición. La estructura condicional compara una variable contra otro(s) valor(es), para que con base en el resultado de esta comparación, se siga un curso de acción dentro del programa. Cabe mencionar que la comparación se puede hacer contra otra variable o contra una constante, según se necesite.

Estas comparaciones utilizan los llamados *operadores relacionales* para verificar el resultado y saber si el resultado es verdadero o falso. Los operadores que podemos utilizar son los siguientes:

```
   > Mayor que
   < Menor que
>= Mayor o igual
<= Menor o igual
   == Igualdad
   ! Diferente
```

TIPOS DE CONDICIONALES

Simples

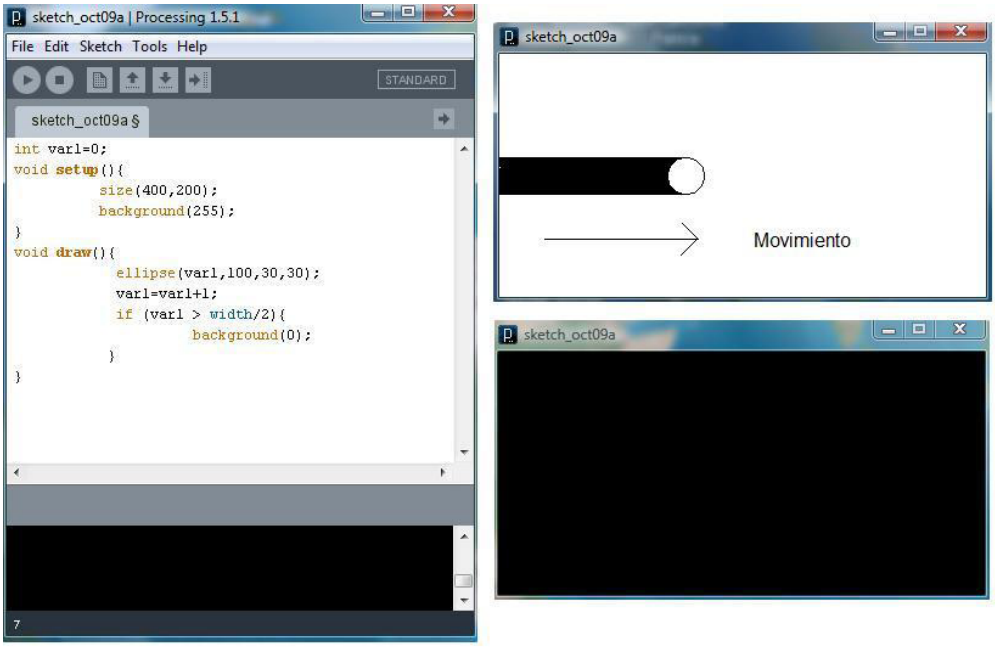
Las estructuras condicionales simples son conocidas como *tomas de decisión. Estas tomas de decisión lo único que hacen es verificar una única comparación. Si el resultado es verdadero, entonces se ejecutará el conjunto de instrucciones que esté dentro del bloque. Si el resultado es falso, simplemente se ignora todo el bloque. El flujo del programa continúa después de hacer esta simple comparación. La estructura simple es:*

```
if (expresión booleana){
// Este conjunto de instrucciones se ejecutan
            //si la condición es verdadera
      }
```

Resulta importante, cuando empezamos a utilizar estructuras de control múltiples, el poder seguir algunas reglas de estilo. Estas reglas nos ayudarán a comprender mejor el programa. En el ejemplo anterior, podemos observar que existe una indentación importante que denota el conjunto de instrucciones que están dentro de la estructura del if.

Otro elemento que podemos observar es la utilización del doble *slash (//).* Este slash le indica a Processing que todo lo que esté posterior al él (en la misma línea) debe de ser ignorado. Esto sirve para agregar comentarios internos en el programa y poder hacerlo más legible a cualquier otro usuario que interactúe con el mismo. Del mismo modo, podemos utilizar el (/*) y el (*/) para definir todo un bloque que queramos sea ignorado por Processing.

A continuación, se muestra un ejemplo muy simple acerca de la utilización de la estructura condicional simple.



En este ejemplo, dentro del programa solo hay un `if`. El programa verificará el valor de la variable y lo comparará con el `valor width/2` (que corresponde al centro de la pantalla). Cuando el valor de `var1` es menor (en este caso, su valor inicial es 0) a la mitad de la pantalla, el resultado de la evaluación es falso y no se ejecutarán las instrucciones dentro del `if`. Cuando el valor de `var1` es mayor a la mitad del ancho de la pantalla, entonces automáticamente se ejecutarán las instrucciones que están dentro del `if`, en este caso el `background(0)`, lo que generará que el fondo de la pantalla adquiera el color negro.

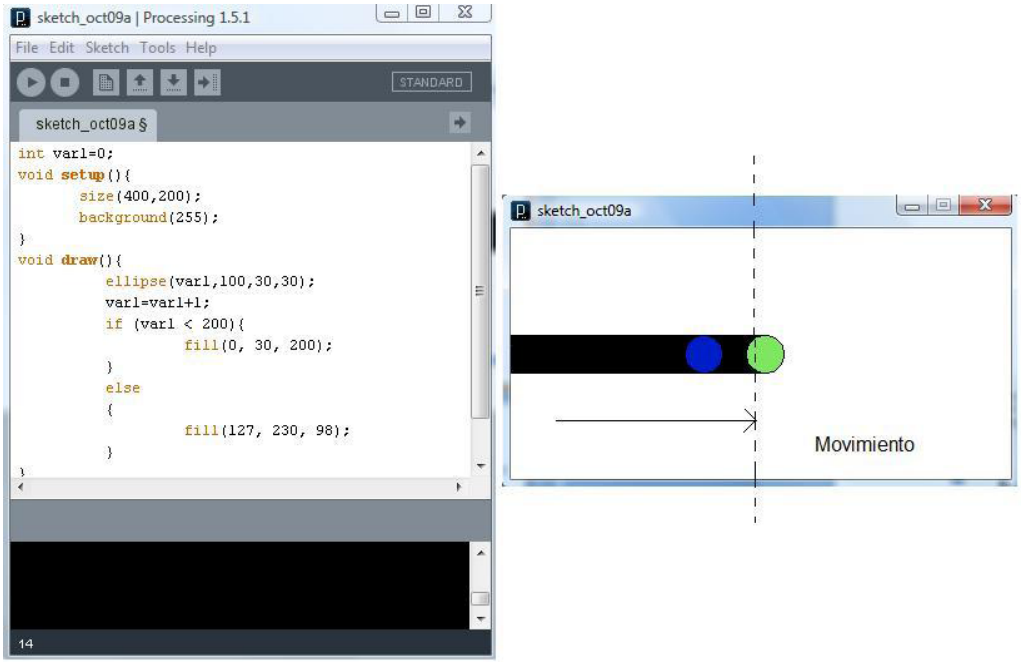
Dobles

Las estructuras condicionales dobles permiten elegir entre dos opciones o alternativas posibles en función del cumplimiento o no de una determinada

condición. Si el resultado de la comparación es verdadero, entonces ejecutará un conjunto de instrucciones, y si el resultado es falso, entonces ejecutará otro conjunto diferente de instrucciones. Una vez realizado este, se continuará con la ejecución del programa.

```
if (expresión booleana){
/* El código que se ejecuta si la condición es verdadera*/
} else {
// El código que se ejecuta si la condición es falsa
}
```

A continuación, se muestra un ejemplo de su funcionamiento.



En este ejemplo, cuando la variable `var1` sea menor a 200, entonces la pelota se pintará de color azul, pero cuando la variable `var1` obtenga el valor de 200, entonces se pintará automáticamente de color verde. Esta estructura tiene únicamente dos condiciones a verificar y está manejado por la estructura `if` y su correspondiente else.

Múltiples

Las estructuras de comparación múltiples son tomas de decisión especializadas que permiten comparar una variable contra distintos posibles resultados, ejecutando para cada caso una serie de instrucciones específicas. La forma común es la siguiente:

```
if (expresión booleana # 1){
    // El código que se ejecuta si la condición # 1 es verdadera
} else if (expresión booleana # 2){
    // El código que se ejecuta si la condición # 2 es verdadera
} else if (expresión booleana # n){
    // El código que se ejecuta si la condición # n es verdadera
} else {
    // Ejecución si ninguna de las condiciones anteriores fueron
    // verdaderas
}
```

Para evaluar múltiples condiciones, utilizamos el else if. Las condicionales son evaluadas en el orden presentadas. Cuando una expresión es verdadera, se ejecuta el código de manera inmediata y todo lo demás es ignorado.

Esta estructura es muy útil cuando tenemos múltiples opciones y lo que se debe de evaluar es la misma variable. Podemos incluir cuantos else if queramos. La ventaja es que cuando se cumpla la condición, automáticamente se ignorarán todas las comparaciones subsecuentes.

A continuación, se muestra un ejemplo donde la pelota cambia de color al verificar la variable var1 en los puntos 100, 200 y 300, todo utilizando la estructura else if.

sketch_oct09a | Processing 1.5.1

File Edit Sketch Tools Help

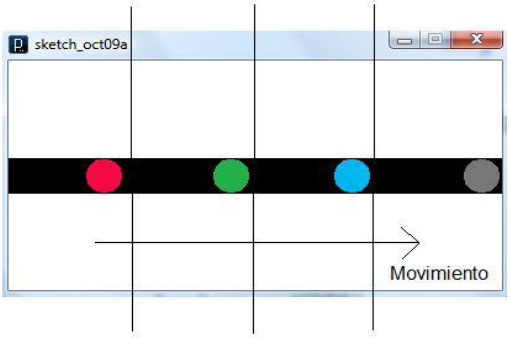
STANDARD

sketch_oct09a §

```
int var1=0;
void setup(){
       size(400,200);
       background(255);
}
void draw(){

          ellipse(var1,100,30,30);
          var1=var1+1;
          if (var1 < 100){
                   fill(250,5,69);
          }
          else if (var1<200)
          {
                   fill(60,250,30);
          }
          else if (var1<300)
          {
                   fill(23,42,250);
          }
          else
          {
                   fill(125,125,125);
          }
}
```

23



Ejercicio propuesto

Utilizando la estructura de las condicionales, crea un rebote en la pelota de arriba para abajo. Cada vez que toque una de las paredes de la ventana, deberá de cambiar de color aleatoriamente. Cuando haya realizado diez rebotes, la pelota deberá regresar al centro; en ese momento, debe finalizar el programa.

CONDICIONALES COMPUESTAS Y LOS OPERADORES LÓGICOS

Hasta ahora en los múltiples ejemplos de condicionales, se han evaluado exclusivamente condiciones simples para poder aplicar o no la estructura

de la condicional. Sin embargo, en algunas de las ocasiones necesitaremos comparar un conjunto de condiciones (dos o más) para verificar si una acción se ejecuta o no.

Para solucionar este problema, tendríamos dos posibles soluciones:

- Generar un conjunto de condiciones independientes disyuntivas, las cuales se verificaría una a una, y donde la acción a ejecutar sería siempre la misma, por ejemplo:

```
if (variable1 == 1){
      variable2=100;
}

if (variable1 == 2){
      variable2=100;
}

if (variable1 == 3){
      variable2=100;
}
```

- Generar un llamado *if anidado para verificar un conjunto de condiciones conjuntivas; así, cuando se cumpla una de las condiciones, entonces se verificará otra condición. Si ambas condiciones son verdaderas, entonces finalmente se ejecuta la acción. Un ejemplo de if anidado es el siguiente:*

```
if (variable1 == 1){
           if(variable2 ==5){
                  ellipse(10,10,10,10);
          }
}
```

Para simplificar estas dos estructuras, existen los llamados operadores lógicos, que permiten hacer una evaluación unificada, dado un conjunto de condiciones. Los operadores lógicos utilizados son:

```
||  (O lógico)
&&  (Y lógico)
   ! (NOT lógico)
```

Al utilizar estos operadores lógicos, la estructura de las condicionales sería de la siguiente manera:

Para el | | (O lógico):

```
if (variable1==1 || variable1 ==2 || variable1==3){
variable2=100;
}
Para el && (Y lógico):

if (variable1==1 && variable2==5){
      ellipse(10,10,10,10);
}
```

De esta manera, se puede apreciar que la condicional es más concisa y se entiende mejor, además de que no se tiene la necesidad de repetir instrucciones.

La evaluación de un conjunto de condiciones simultáneas requiere que, al final de la verificación global, se tenga un veredicto para ejecutar, o no, un bloque de código. Para estos casos, se deberán de seguir las llamadas *tablas de verdad, para finalmente poder verificar si un conjunto de condiciones son verdaderas o no.*

El &&. La conjunción es un operador que opera sobre dos valores de verdad. Típicamente los valores de verdad de dos condiciones, devolviendo el valor de verdad verdadero cuando ambas proposiciones son verdaderas, y falso en cualquier otro caso. Es decir que es verdadera cuando ambas son verdaderas.

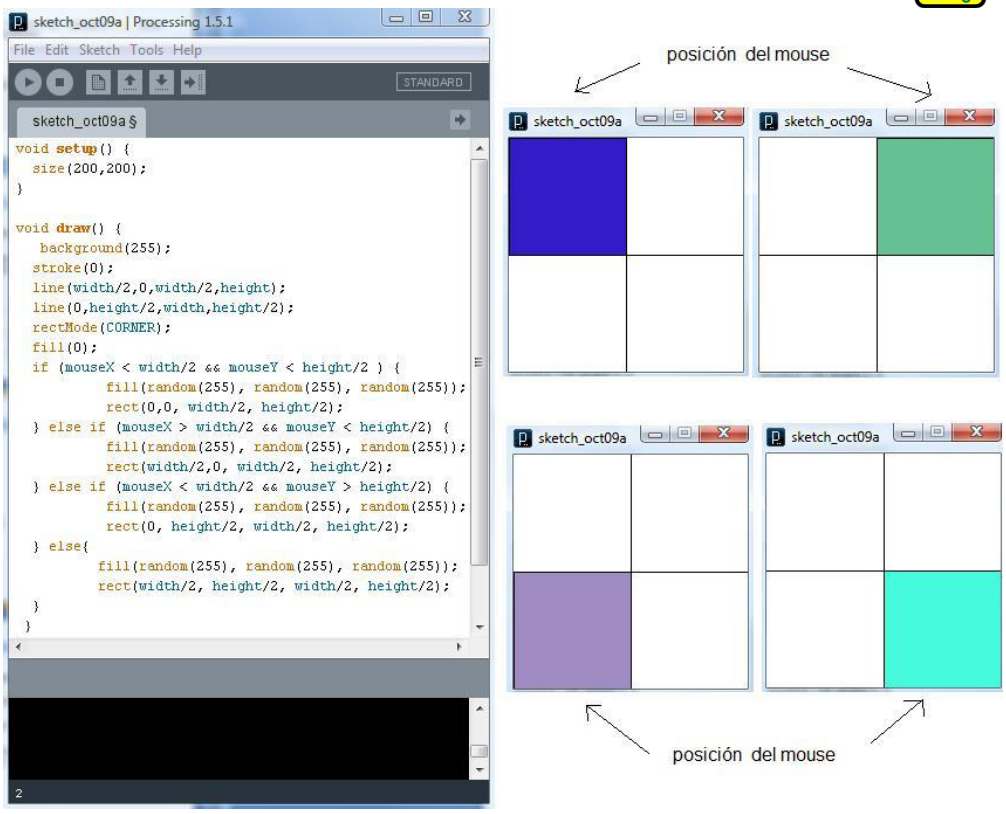
La tabla de verdad de la conjunción es la siguiente:

```
V && V -> V
V && F -> F
F && V -> F
F && F -> F
```

El ||. La disyunción es un operador que funciona sobre dos valores de verdad, típicamente los valores de verdad de dos condiciones, devolviendo el valor de verdadero cuando una de las proposiciones es verdadera, o cuando ambas lo son, y falso cuando ambas son falsas.

```
V | | V -> V
V | | F -> V
F | | V -> V
F | | F -> F
```

A continuación, se muestra un ejemplo del uso de estas condicionales compuestas. En este ejemplo, podemos apreciar el funcionamiento del operador lógico `&&`. En la estructura del `if` se realizan en un par de comparaciones, al igual que en los `else if`. En este caso, se ubica la posición del mouse para determinar en qué sector se va a dibujar un rectángulo de color aleatorio.

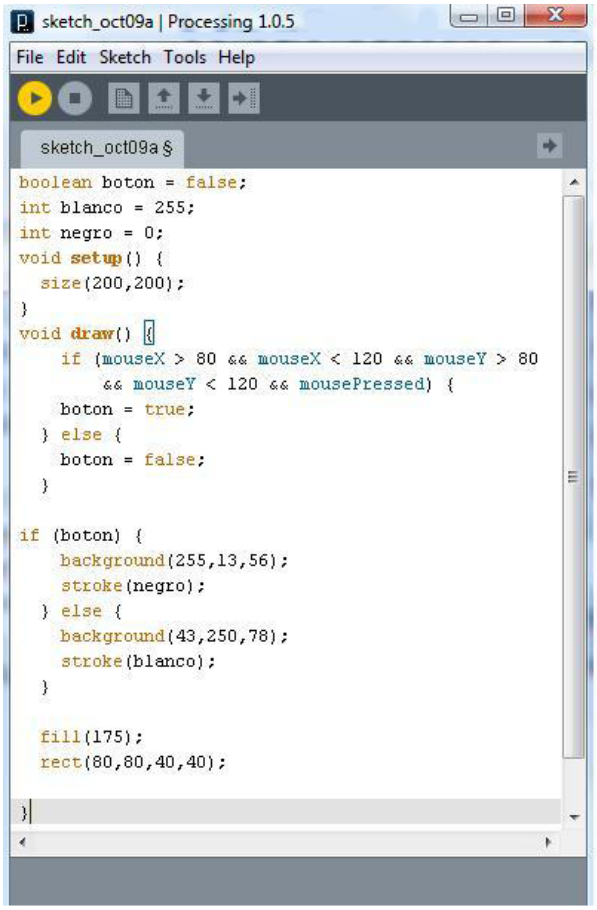


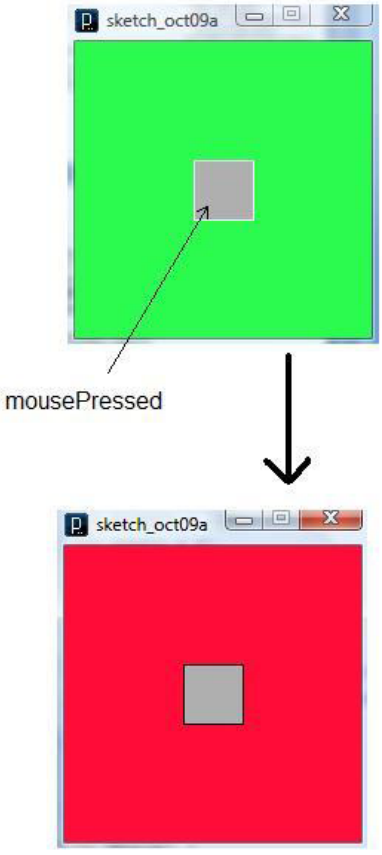
CONSTRUCCIÓN DE BOTONES

Los botones, en computación, son una metáfora utilizada en las interfaces gráficas para simular el funcionamiento de un botón corriente. Los botones suelen representarse como rectángulos con una leyenda o ícono dentro. La principal funcionalidad de los botones es realizar una selección. Esta selec-

ción se realiza mediante dos estados que puede tener el botón (encendido o apagado). Con esta primicia, las variables booleanas pueden ser de gran ayuda. Recordemos que las variables booleanas solo aceptan un par de valores (true y false).

A continuación, se muestra un ejemplo de la elaboración de un botón.

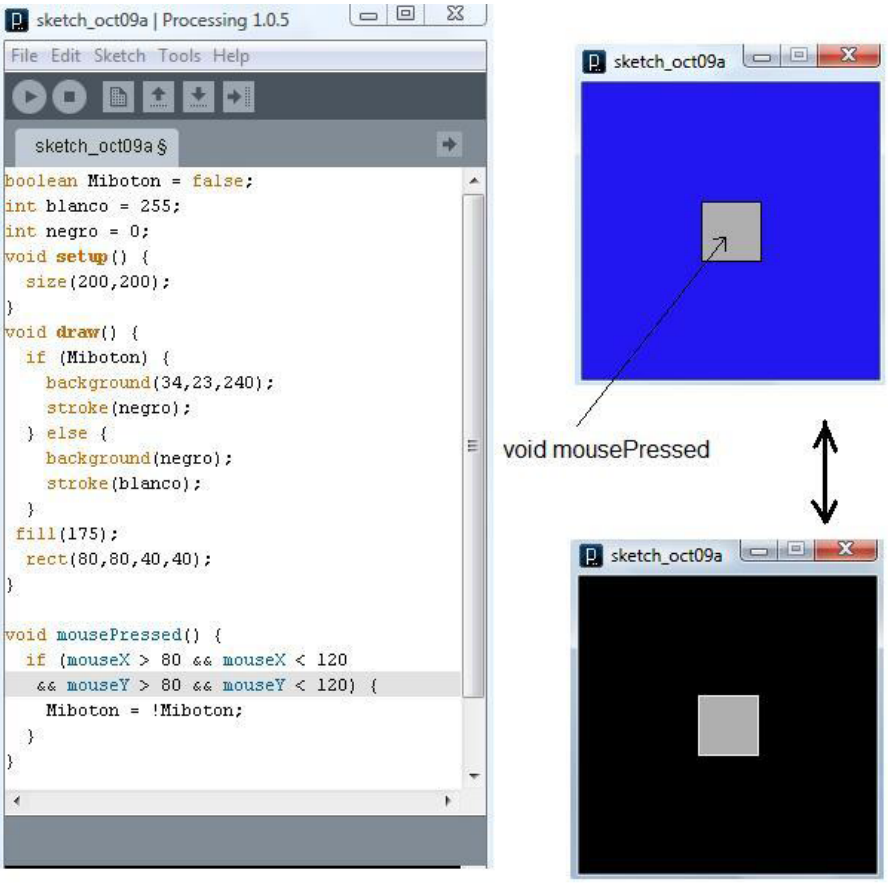




En este ejemplo, primero se define `botón` como una variable booleana y se asigna el valor de falso. Posteriormente, se debe verificar la posición del mouse en donde se quiere ubicar nuestro botón y delimitar con las coorde-

nadas `x` y `y`. Además, es necesario verificar que la variable `mousePressed` esté activa. Si toda esta condición se cumple, entonces daremos el valor de `true` a nuestro botón. En el segundo bloque del `if`, únicamente se verifica el estado del botón. Si es true, entonces asignamos un color rojo al fondo de la pantalla, y si es false, entonces el fondo se quedará en verde. Cabe hacer notar que, para activar el botón, el mouse debe de estar siempre presionado.

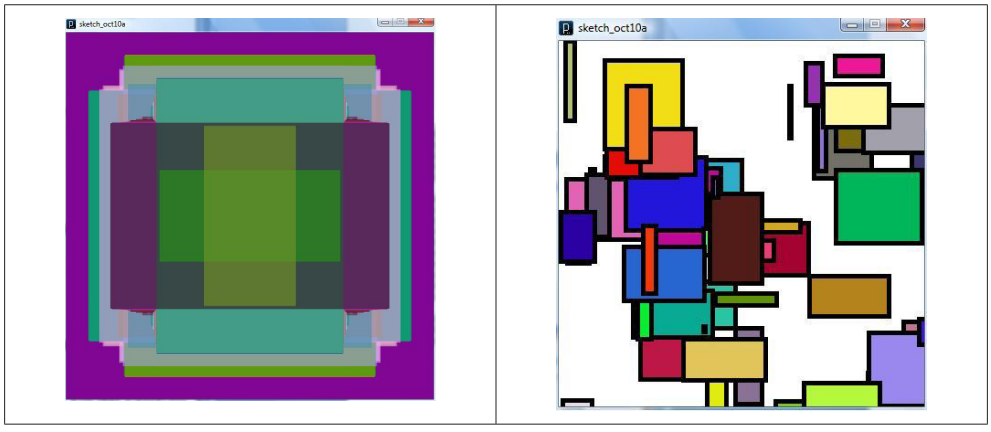
A continuación, se muestra una manera alternativa –y probablemente más útil– de programar botones en Processing.



Este ejemplo funciona de manera un poco diferente al anterior. Al oprimir el botón, provoca que este quede activo todo el tiempo, y se tendrá que desactivar hasta volver a dar clic una vez más en el botón. En este ejemplo, se definió la función `mousePressed` de manera separada. Al hacer un clic, se entra a la función `mousePressed` y se verifican las coordenadas del mouse. Si se está dentro de las coordenadas definidas, entonces se debe de cambiar el valor a `Miboton`, si estaba con el valor de `false`, ahora será `true` y viceversa. El `if` que se encuentra en la función `draw` lo único que hace es verificar el estado del botón, y si es verdadero, entonces dibuja el fondo de pantalla de color azul. En caso contrario, lo dibuja de color negro.

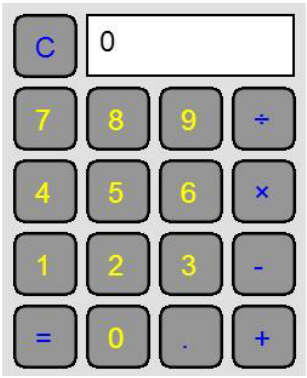
Actividad integradora 1

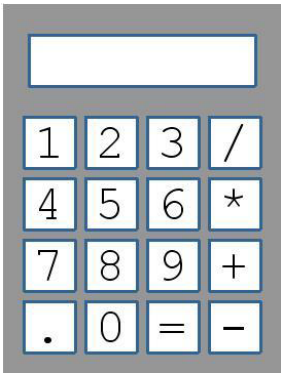
Utilizando las condicionales y variables, crea un programa que genere un dibujo libre y completamente aleatorio utilizando las formas geométricas de base que se revisaron hasta el momento.



Actividad integradora 2

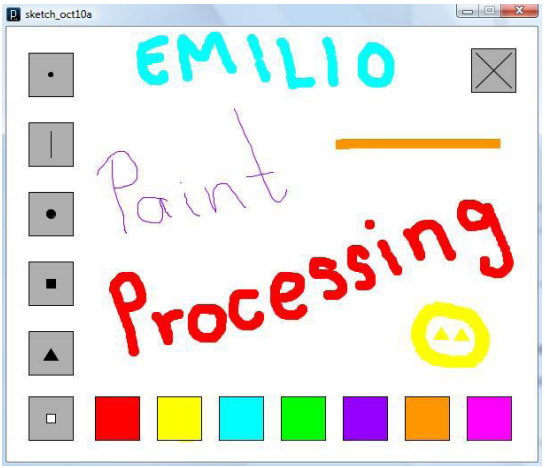
Utilizando los botones que se revisaron en el capítulo, diseña una calculadora básica que permita realizar las operaciones básicas (+,-,/,*). El ingreso de los números será mediante los botones y el despegado del resultado deberá ser directamente en la consola.



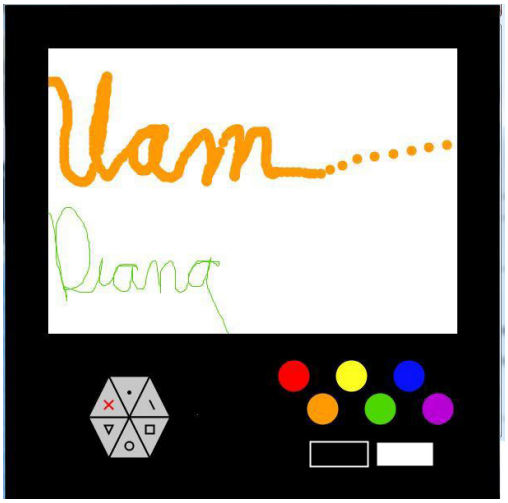


Experiencias previas

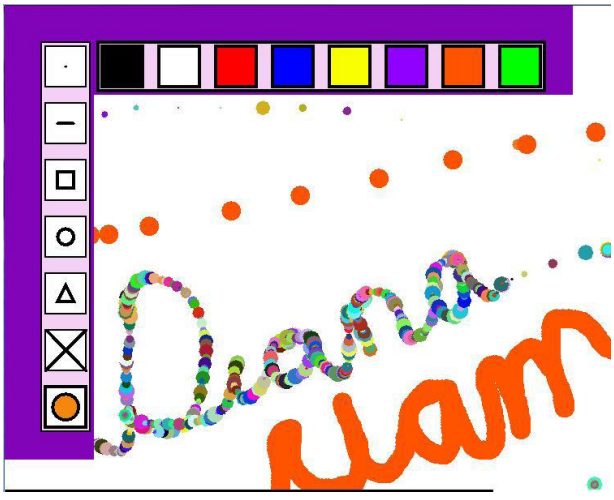
Uso de botones, variables, colores y formas para generar un miniPaint.



Autor: Emilio Trasviña Díaz
Ciencias de la Comunicación
Trimestre 13 Invierno



Autor: Diana Noemí Aguilar González
Ciencias de la Comunicación
Trimestre 13 Invierno



Autor: Diana Flores Chávez
Ciencias de la Comunicación
Trimestre 13 Invierno

Capítulo 6. Iteración

"El optimismo es un riesgo laboral de la programación;
el *feedback* es el tratamiento"
Kent Beck

Objetivo

- Conocer, comprender y aplicar las estructuras de iteración.
- Entender la forma de trabajar del `while` y del `for` en la resolución de problemas.

Actividades de aprendizaje

- Identificar la necesidad de utilizar las estructuras iterativas.
- Estudiar los operadores de incremento y decremento.
- Realizar ejemplos aplicando las iteraciones.

La iteración

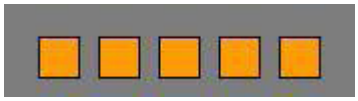
En la vida diaria existen situaciones que frecuentemente se resuelven por medio de realizar una secuencia de pasos que puede repetirse muchas veces. Por ejemplo:

- El proceso que seguimos para comer, mientras no terminemos la comida.
- El proceso de pagar con monedas en el estacionamiento hasta llegar a la cantidad necesaria.
- Las operaciones que realizamos para llamar por teléfono, mientras no se logre la comunicación.
- Las acciones que hacemos mientras estamos en el autobús.

Estos son algoritmos de la vida cotidiana, pero tienen la particularidad de que la ejecución de alguno de sus pasos puede repetirse muchas veces, mientras no se logre la meta trazada. A este tipo de algoritmo se le conoce como algoritmos iterativos o repetitivos.

Supongamos que nos piden que realicemos el dibujo de cinco rectángulos de las mismas características pero en posición diferente. En realidad, no tendríamos problemas para escribir cinco veces la función rectángulo.

```
rect(20,20,20,20);
rect(50,20,20,20);
rect(80,20,20,20);
rect(110,20,20,20);
rect(140,20,20,20);
```



Básicamente, este programa cumple con el objetivo. Pero, ¿qué podemos apreciar? Se repitió cinco veces la misma instrucción. Ahora, ¿qué pasaría si quisiéramos dibujar no solo cinco sino cien rectángulos? Tendríamos que agregar más líneas de código y seguramente tardaríamos mucho tiempo en escribirlo. Sin embargo, existe un recurso para solucionar este tipo de problemas, nos referimos a las estructuras de iteración. Una iteración consiste en una repetición de un bloque de sentencias un número determinado de veces o hasta que se cumpla una condición. De esta manera, el código puede simplificarse.

LA INSTRUCCIÓN WHILE

Esta instrucción, permite repetir un número indeterminado de veces una instrucción o un bloque de instrucciones hasta que no se cumpla una condición específica. La estructura del `while es la siguiente:`

```
while (expresión booleana){
      // Este es el conjunto de instrucciones que se ejecutan
     // si la condición es verdadera
}
```

Al cumplirse la expresión booleana, el flujo del programa entrará a la estructura while y repetirá la ejecución de las instrucciones de manera indefinida

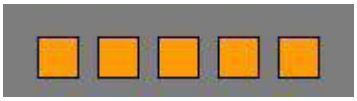
hasta que la condición no se cumple. En ese momento, el flujo del programa continuará con su ejecución normal.

Los elementos que deben de estar integrados para que la estructura while pueda trabajar de manera correcta son:

- La inicialización: establece un valor inicial para aquellas variables que participan en la condición booleana.
- La condición: es la expresión booleana que se evalúa como verdadero o falso, según el valor de las variables que participan en esa condición con la cual se decide si el cuerpo del ciclo debe repetirse o no.
- La actualización: es una instrucción que debe de estar en el cuerpo de la estructura `while`, la cual hace cambiar el valor de las variables que forman parte de la condición. Dentro de la estructura del `while` se debe de actualizar la condición. No necesariamente la actualización se realiza en una sola acción, puede darse en varias acciones y ser tan compleja como sea necesario, pero siempre dentro de la estructura.

Entonces, para nuestro ejemplo, integrando la estructura `while` el resultado es el siguiente:

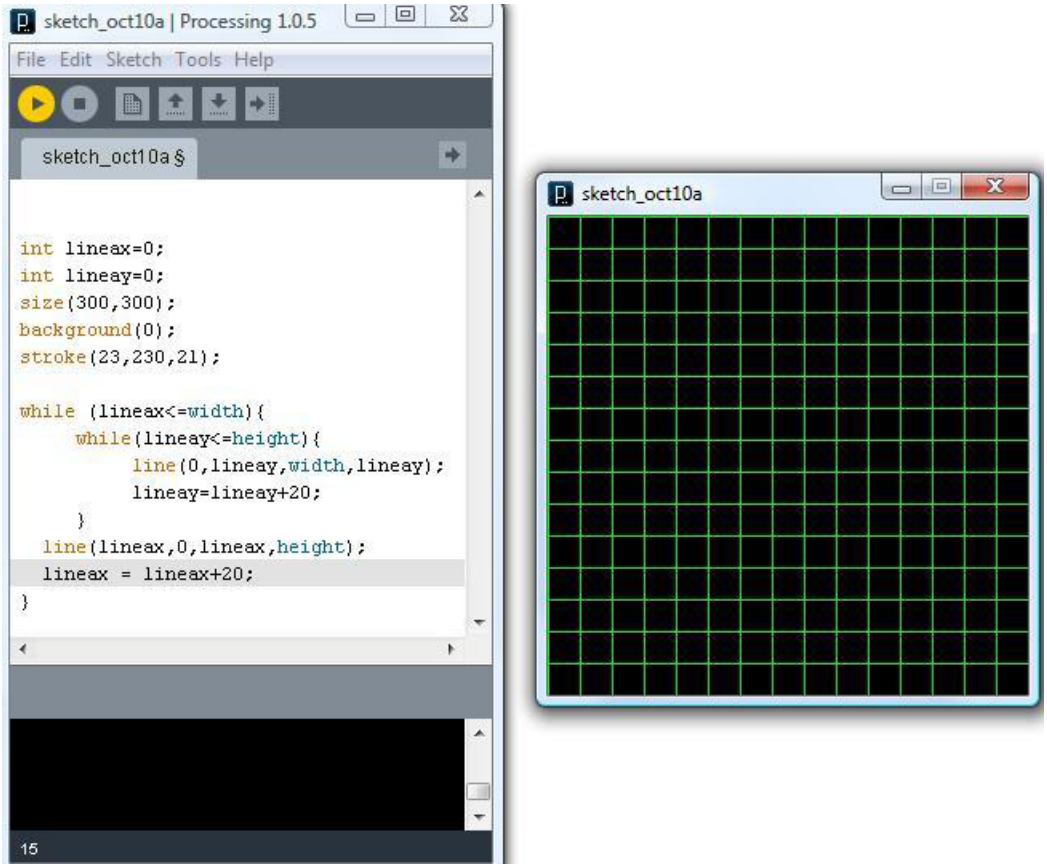
```
int variable1=20;
while(variable1<=140){
       rect(variable1,20,20,20);
       variable1 = variable1 + 30;
}
```



Los elementos integrados en la estructura son representados de la manera siguiente. La inicialización se debe definir mediante la asignación inicial del valor de la `variable1`. En este caso `variable1=20`. Es el punto de partida de nuestra iteración. La condición `(variable1<=40)` es la que verifica la repetición constante del bloque de instrucciones. Finalmente, la actualización `(variable1=variable1+30)` es la que se encarga de hacer evolucionar la variable1 que a su vez será comparada cada vez mediante la condición. En el ejemplo se repetirá la acción hasta que la `variable1` sea mayor a 40. En este caso, se terminará con la ejecución de la iteración y el programa seguirá su flujo normal.

Es importante mencionar que esta estructura podría no ejecutarse, si es que la condición definida nunca se cumple. Otro caso específico que puede ocurrir es que el ciclo sea infinito, esto quiere decir que la condición siempre se cumple.

Al igual que la estructura if, al utilizar la estructura while también podemos utilizar otra estructura while dentro. A esto se le llama un ciclo anidado. A continuación, se muestra un ejemplo de la construcción de una malla en la pantalla utilizando while anidado.



En este ejemplo, se aprecia que existen dos ciclos `while`, uno dentro del otro. Es importante mencionar que tanto las condiciones como las variables que se deben de verificar tienen que ser diferentes, con el propósito de evitar conflictos entre las variables.

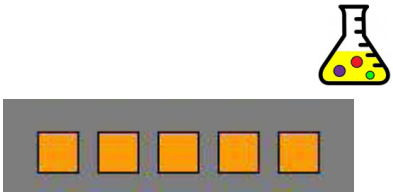
LA INSTRUCCIÓN FOR

La instrucción `for` es utilizada cuando sabemos de antemano el número de veces que debemos repetir un conjunto de instrucciones. También es un bloque, y está definido de la siguiente manera:

```
for (inicialización, condición, actualización)
{
   // Este es el conjunto de instrucciones que se ejecutan
   // n número de veces
}
```

La estructura del for debe de tener los tres elementos de la iteración (inicialización, actualización y condición), representadas de manera implícita en la propia estructura iterativa.

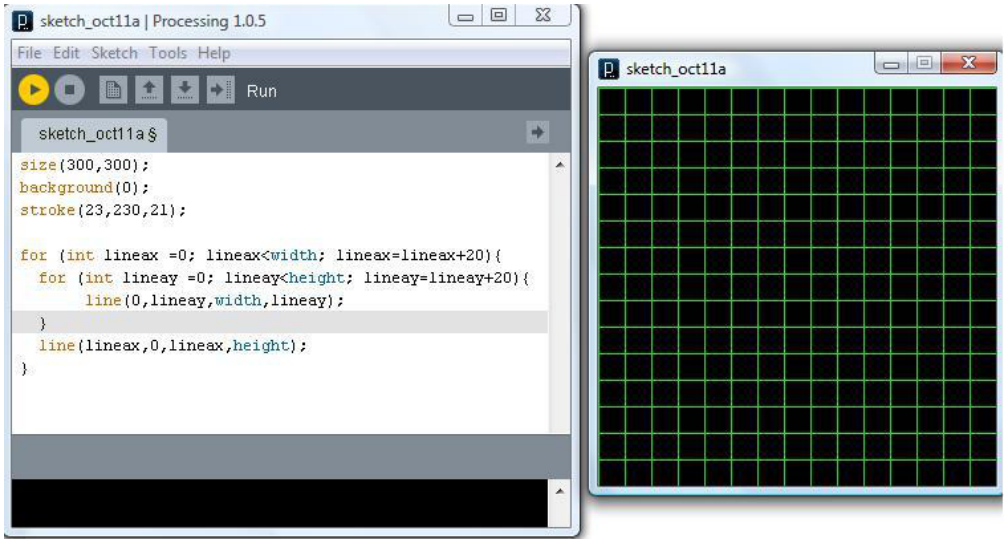
Así, para el ejemplo de dibujar un número n de rectángulos, la estructura for sería la siguiente:



```
for(int variable1=20; variable1<=140;
    variable1=variable1+30)
{
    rect(variable1,20,20,20);

}
```

La estructura del `while` y del for son equivalentes, la diferencia es que en el `for` concentramos la inicialización, la condición y la actualización en la definición de la estructura, mientras que en el `while` esta definición es más libre. Al igual que en la estructura del `while`, en el ciclo `for` también podemos encontrar anidamientos, es decir, un `for` dentro de otro `for`. A continuación, se muestra el mismo ejemplo de la construcción de la malla, pero ahora utilizando el for anidado.

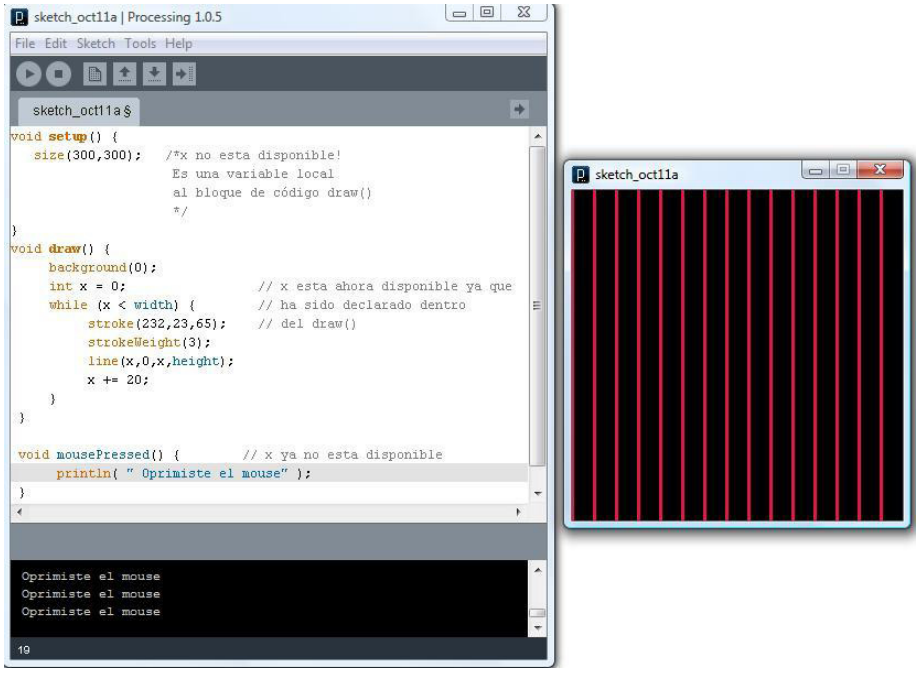


ES BUENO SABER QUE ...

- Existen los llamados operadores de incremento y decremento, los cuales pueden verse como unos atajos, pueden hacer el código mucho más fácil de escribir y de leer. Es menos complicado entender una línea de código con pocas letras.
- Los operadores de incremento y decremento son de los mejores atajos. Se utilizan a menudo para modificar lass variables que controlan el número de veces que se ejecuta alguna instrucción dentro de un ciclo.
- El operador de decremento es -- , que significa *decrementar una unidad.* El operador de incremento es ++ , que significa incrementar una unidad. Los operadores de incremento y decremento producen el valor de la variable como resultado. Algunos ejemplo son:

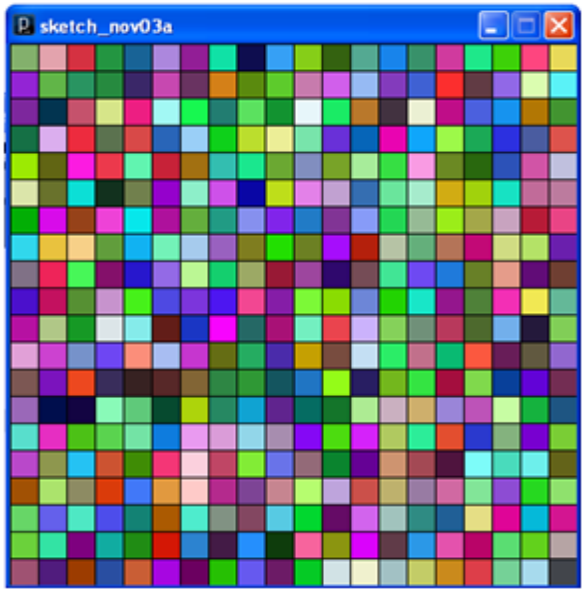
```
• x++; es equivalente a: x = x+1;
• x--; es equivalente a: x = x-1;
• x+=2; es equivalente a x = x+2;
• x*=3; es equivalente a x = x*3;
```

- Todos los programas que se han trabajado, utilizan las estructuras `setup()` y `draw()`. Una de las características de esta segunda función es la de siempre ejecutar un ciclo y estar indefinidamente repitiéndose. Sin saberlo, utilizamos ya una estructura cíclica. Al introducir un ciclo `for` o `while`, estamos provocando la generación interna de un ciclo anidado.
- Existen en la práctica dos tipos de variables: las globales y las locales. Hasta el momento, siempre hemos declarado las variables al inicio del programa, pero en la práctica pueden ser declaradas en cualquier lugar del programa. Las variables globales son aquellas que se declaran en la cabecera del programa (fuera del `setup()` y del `draw()`) y podrán ser utilizadas durante todo el programa. Las variables locales son declaradas dentro de algún bloque de código (`setup()`, `draw()`, `mousePressed()`, `keyPressed()`, `if`, `while`, `for`) y únicamente pueden ser utilizadas dentro del bloque. Ahora bien, la pregunta obligada es: ¿para qué usamos las variables locales? ¿No es más fácil declarar al inicio todas las variables? En la práctica, es más fácil, eficiente, práctico y menos confuso declarar las variables cuando las utilizamos. A continuación, se muestra un ejemplo del uso de una variable local.



Actividad integradora

Realiza la generación automática de una cuadrícula utilizando los colores `random`. Debes utilizar una sola instrucción `rect()` para poder dibujar todo el conjunto de rectángulos. La cuadrícula se debe parecer a esto:



Intenta hacerlo utilizando:

un `for` dentro de otro `for`
un `for` dentro de un `while`
un `while` dentro de un `while`
un `while` dentro de un `for`
un `while` dentro del `draw`
un `for` dentro del `draw`

Experiencias previas

Autor: Rosalba González
Lic. Diseño
Trimestre 12 Invierno

Conclusión

Este material es muy interesante para los alumnos que no han tenido una experiencia previa con los lenguajes de programación. Incluso para aquellos que han tenido experiencias malas al haberse enfrentado por primera vez con lenguajes de programación que no están adaptados para un primer acercamiento con el maravilloso mundo de la programación.

En este material, hemos estudiado y experimentado con los elementos fundamentales de toda Programación Estructurada (la secuencia, la condición y la iteración). Un programa escrito con este enfoque tendrá una estructura clara, fácil y comprensible.

Hemos utilizado el lenguaje de programación Processing para plasmar de manera visual el resultado de estos elementos. Hemos trabajado con formas, colores, estructuras, además de promover la creatividad y la expresión artística.

Processing permite, además de lo explorado en este material, trabajar con muchos otros elementos. Por ejemplo:

- La representación 3D, donde se estudia el espacio no solo en dos dimensiones, sino en tres.
- La manipulación de texto gráfico, que permite generar múltiples efectos con formas y colores al texto.
- La manipulación de imágenes, donde no solo es posible desplegar, sino manipular las imágenes pixel por pixel.
- La manipulación de video y sonido, agregando efectos y creaciones tanto sonoras como visuales.
- La manipulación de cadenas y archivos para profundizar en el procesamiento de caracteres y manejo de archivos.
- Manejo de cámara, luces y sombras para una mejor representación.

- Emplea múltiples librerías que permiten utilizar instrucciones adicionales para trabajos específicos con tipografía, matemáticas, redes, simulaciones, visualizaciones, etc.

Si se desea profundizar en algunos de estos elementos en el sitio http://processing.org hay un gran número de recursos que sirven para generar aplicaciones más elaboradas.

La experiencia de impartir la UEA Fundamentos de Programación Estructurada a alumnos que no son de un área de ingeniería o que no pertenecen a una licenciatura donde es un curso obligatorio, es gratificante. En nuestra experiencia, hemos sido capaces de explicar de manera fácil cada uno de los conceptos, de dejar actividades que incitan a la creatividad y, sobre todo, hemos visto cómo los alumnos logran programar, a diferencia de haber implementado un método tradicional con los ya conocidos lenguajes de programación.

Al final del curso, el alumno está capacitado para usar cualquier otro lenguaje de programación, debido a que los fundamentos explicados en la clase (y en este material didáctico) son aplicables a cualquier lenguaje.

Este curso tuvo como uno de sus propósitos, abrir las puertas hacia el apasionante mundo de la programación.

Bibliografía

Bohnacker, H. *et al*. (2012). *Generative Design: Visualize, Program, and Create with Processing*. Nueva York_ Princeton Architectural Press.
Glassner, A. (2010). *Processing for Visual Artists: How to Create Expressive Images and Interactive Art*. Natick, MA: AK Peters, Ltd.
Greenberg, I. (2007). *Processing: creative coding and computational art*. Nueva York: Apress.
Reas, Casey & Fry, B. (2007). *Processing: a programming handbook for visual designers and artists*. Cambridge: MIT Press.
Reas, Casey & Fry, B. (2010). *Getting Started with Processing*. California, EUA: O'Reilly Media, Inc.
Shiffman, D. (2009). *Learning Processing: a beginner's guide to programming images, animation, and interaction*. Massachusetts: Morgan Kaufmann,.
Terzidis, K. (2009). *Algorithms for visual design using the processing language*. Indianapolis: John Wiley & Sons.

Recursos web*:

http://processing.org
http://www.joan.cat/processing/cs/
http://www.learningprocessing.com/tutorials/

Recursos Youtube*:

https://www.youtube.com/watch?v=9UcL8B0GQuE&list=PL374E-5107CB62B2BE
https://www.youtube.com/watch?v=ZByppaYSJB8

* Todos los recursos web se encuentran también en el sitio móvil: Sitio móvil de Hola mundo con Processing: http://winksite.com/aranike/hmp

Sitio móvil de Hola Mundo con Processing

El objetivo del sitio móvil es contar con recursos digitales que puedan ser compartidos con el grupo. Esta iniciativa fue implementada a partir del trimestre 14 Otoño.

Ejemplo de la sección de videos

Glosario

Algoritmo. Conjunto de pasos ordenados para lograr un objetivo.

Argumentos. Variable que puede ser recibida por una rutina o función.

Botón. Metáfora común, utilizada en interfaces gráficas con objetivo similar al de un botón corriente.

Conjunción. Operador que devuelve el valor de verdad verdadero cuando ambas proposiciones son verdaderas.

Condición. Sentencia que se puede evaluar como verdadera o falsa.

DFD. Diagrama de Flujo de Datos.

Disyunción. Operador que devuelve el valor de verdad verdadero cuando alguna proposición es verdadera.

HSB. Combinación de colores (Hue, Saturation, Brightness).

Instrucción. Conjunto de datos insertados en una secuencia estructurada o específica que el procesador interpreta y ejecuta.

Interfaz. Conexión física y funcional entre dos sistemas o dispositivos de cualquier tipo.

Iteración. Acto de repetir un proceso con el objetivo de alcanzar una meta deseada.

Lenguajes de programación. Es un lenguaje formal diseñado para expresar procesos que pueden ser llevados a cabo por las computadoras. Ejemplos: Java, Processing, Pascal, C, C++, Lisp, Fortran, Python, MatLab, etc.

MBR. Minimum Bounding Rectangle (Rectángulo Mínimo de Delimitación).

Operadores incremento y decremento. Símbolos que sirven para aumentar o disminuir el valor de una variable de manera automática (++, --).

Operadores lógicos. Composición lógica que nos proporciona un resultado a partir de que se cumpla o no cierta condición.

Operadores relacionales. Símbolos que se usan para comparar dos valores (<,>,==,!=).

Pixel. Picture element. Es la menor unidad homogénea en color que forma parte de una imagen digital.

Programa. Es una secuencia de instrucciones escritas para realizar una tarea específica con una computadora.

Random. Proceso cuyo resultado no es previsible más que en razón de la intervención del azar.

RGB. Combinación de colores (Red, Green, Blue).

Sintaxis. Forma visible de un lenguaje de programación que describe las combinaciones posibles de los símbolos que forman un programa sintácticamente correcto.

Variables. Espacio en el sistema de almacenaje (memoria principal de una computadora) y un nombre simbólico que está asociado a dicho espacio.

Hola Mundo con Processing se terminó de imprimir en agosto de 2015 de forma digital en los talleres de Imprenta 1200+ Andorra 29. Colonia Del Carmen Zacahuitzco, México D.F. Tel. (52)55218493.

El tiraje consta de 100 ejemplares de 17x24 cm, 90 páginas cada uno, a cuatro tintas, encuadernación pegado cubierta flexible. En su composición se utilizaron las familias Avenir y Courier New. Se empleó papel reciclado de 90g para páginas interiores.