Programas más complejos

Para recapitular, es importante clarificar lo que se ha generado dentro del ambiente de programación:

1. Se pueden generar dibujos que en realidad son instrucciones para la computadora que ya tiene las rutinas
2. Se puede pedir ingreso de valores de variables por parte del usuario para hacer más “dinámica” l respuesta en programación
3. Las variables son la base de un código que puede actuar de manera distinta (pocos cambios, funcionalidad distinta)
4. Las constantes son los valores que las variables tienen y son los elementos últimos de definición
5. Para hacer una buena rutina de programación, es necesaria la planeación y análisis antes de la implementación

Condicionales

En ocasiones se querrán introducir vertientes al programa que puedan, dependiendo de la entrada del usuario, generar un procedimiento u otro. Este tipo de comportamiento requiere el uso de condiciones para la decisión. Estas condiciones siguen la teoría englobada dentro de las matemáticas discretas, rama de la programación que está orientada al estudio de números enteros, conjuntos, álgebra boleana, inducción matemática, etc.

Antes de identificar las condiciones, es necesario recuperar algunos conceptos de teorías de conjuntos

---

–

Teoría de conjuntos, funciones, restricción de funciones, y álgebra boleana

El área de las matemáticas que estudia la separación en “bolsas” de elementos como lo pueden ser números o simplemente “objetos” es la teoría de conjuntos. Esta subespecialidad está particularmente relacionada con la probabilidad y la estadística, de manera que basado en datos conocidos (estadística) se puede aventurar una aproximación precisa sobre la selección de un elemento del conjunto. De manera análoga, la teoría de conjuntos, la probabilidad y la estadística, permitiría conocer con un cierto grado de seguridad el siguiente número que pudiera ser generado en una lotería, o el resultado que podría existir en un partido de un deporte.

Aunque estas definiciones parecerían “magia negra”, son en realidad un estudio de propiedades especificas de los números enteros y la repetición de selecciones aleatorias de ellos.

De estas situaciones se ha definido conjunto en el área matemática como una agrupación de elementos definida explícita o implícitamente. Estos elementos normalmente son números, generalmente enteros, tanto positivos como negativos que representan o son la abstracción de algo. La manera en cómo se representan en las matemáticas es la siguiente:

Conjunto definido explícitamente: S={1,2,3,4} donde 1,2,3,4 cuentan de manera independiente como un elemento a ser seleccionado. Pueden ser los identificadores de una persona (IDs), o el número asignado a un carro. Así mismo, podrían ser definidos como S={a,b,c,d,e}. Estas letras podrían tener valores o simplemente ser letras. Los conjuntos normalmente se definen con una mayúscula.

Conjunto definido implícitamente: agrupación que se integra a raíz de una serie de propiedades de los elementos que la comprenden sin necesidad de colocarlos. Un ejemplo es:

Sea S el conjunto definido por {x | 0<x<10 }. Esta línea se puede leer como S incluye todos los números **x** tal que **x** es cualquier número mayor a 0 y menor a 10. De esta manera el conjunto S es {1,2,3,4,5,6,7,8,9}. Probablemente no es claro en este momento por qué esta forma es adecuada. Para ilustrar esto, pensemos en todos los números que pueden ser seleccionados en una rifa del 1 al 999,999. Con esta definición, más allá de colocar todos los números del 1 al 999,999 se define la condición de x|1<=x<=999,999 y se establece un formato más simple.

Operaciones Unión, Intersección, Diferencia

Dado que pueden existir una infinidad de conjuntos en el universo, es posible querer hacer operaciones entre ellos. Particularmente existen tres:

1. La posibilidad de juntar todos los elementos de dos conjuntos: Unión
2. La posibilidad de tener un conjunto solo con los elementos comunes en dos: Intersección
3. La posibilidad de tener un conjunto solo con los elementos que no son comunes en dos: Diferencia

Considerando los siguientes ejemplos:

A = {1,2,3,4} y B= {1,5,6,7}

Unión: incluye todos los elementos de ambos:

A U B = {1,1,2,3,4,5,6,7}. Dado que el elemento 1 es el mismo elemento en ambos conjuntos, es posible colocarlo solo una vez (si los conjuntos fueran números de pasaporte de una persona, la persona forma parte de estas dos listas, por lo que al final es solo una persona).

Intersección: incluye todos los elementos que estén en ambos

A ∩ B = {1}. Dado que el 1 es el único elemento encontrable en ambos conjuntos, es la intersección de los dos.

Diferencia:

A – B = {2,3,4} . La diferencia es “quitar” los elementos comunes de un elemento en el otro. El 1 es el único número que está en B, por lo que se restan los elementos comunes.

En la computación se analizarán estos conjuntos como posibles valores en condiciones y su Unión, intersección y diferencia a raíz de operadores como AND y OR (y sus negaciones).

En muchas ocasiones los conjuntos no solo son números individuales sino pares o tercias o subconjuntos de más números que pueden ser definidos explícita e implícitamente:

Explícita: {(1,2),(6,2),(3,6),(8,9)} que podrían significar los IDs de dos personas que probablemente competirán juntas.

Implícita {(x,y) | 0<x<10 , y = x+1} de esta forma, el conjunto de pares x,y serán todas las x entre 0 y 10 y todas las y que sumen 1 a la x = { (1,2) (2,3) (3,4) (4,5) (5,6) (6,7) (7,8) (8,9) (9,10) }. Las operaciones de unión, intersección y diferencia aplican exactamente igual, contando con que el elemento, para que sea igual, tiene que serlo en todo el par, en el mismo orden. Ejemplo:

A = { (1,1),(2,3),(4,5),(6,5)} y B = {(2,3), (5,6) , (7,7) }

A U B = { (1,1) (2,3) (4,5) (6,5) (5,6) (7,7)}

A ∩ B = { (2,3) }

A - B = { (1,1) (4,5) (6,5) }

En este punto, la teoría de conjuntos se conecta con las relaciones y funciones algebraicas comunes como lo es y = x+1. Esta “función” se puede definir como el conjunto de todas las x elemento del universo de números y las y que son x + 1. El par (3,3) no forma parte de esa definición.

Dado que las funciones son graficables, también los pares o tercias de los conjuntos. Es en este punto donde es importante definir “relaciones” que existen entre los números. Esto conformara la base de las condiciones en los estatutos “if” de la programación. Las relaciones entre dos variables (siempre serán dos) definen los “momentos” en los que un pedazo de código se pueda ejecutar. En métodos más complejos una serie de relaciones binarias conectadas podrán generar diferentes comportamientos.

Nota: explícitamente se está dejando de lado la definición formal de función y relación para mantener este anexo en términos simples, sin embargo, se recomienda la investigación a detalle de estos dos conceptos.

Así, las relaciones importantes entre dos VARIABLES en programación son las siguientes:

< : menor que. La variable del lado izquierdo deber ser menor que la del lado derecho para formar parte del conjunto definido por esta relación, por ende para que un condicional sea verdadero.

> : Mayor que: La variable del lado izquierdo deber ser mayor que la del lado derecho para formar parte del conjunto definido por esta relación, por ende para que un condicional sea verdadero.

<= : Menor o igual que: La variable del lado izquierdo deber ser menor o igual que la del lado derecho para formar parte del conjunto definido por esta relación, por ende para que un condicional sea verdadero.

>= : mayor o igual que: La variable del lado izquierdo deber ser mayor o igual que la del lado derecho para formar parte del conjunto definido por esta relación, por ende para que un condicional sea verdadero.

= : igual que: La variable del lado izquierdo deber ser igual que la del lado derecho para formar parte del conjunto definido por esta relación, por ende para que un condicional sea verdadero.

!= : diferente a: La variable del lado izquierdo deber ser diferente que la del lado derecho para formar parte del conjunto definido por esta relación, por ende para que un condicional sea verdadero.

Estas relaciones entre dos variables, deben ser estudiadas dentro de los condicionales para analizar el conjunto de posibilidades a la hora de acceder a un estatuto condicional. En la programación, dado que no existen otros elementos de comparación, es suficiente entender estos.

Para poder implementar las opciones de unión e intersección de estos conjuntos, la computación introduce el uso de AND y OR que evalúan, dentro de la unión o intersección, la pertenencia de relaciones a una serie de conjuntos.

La operación AND evalúa que una relación entre dos variables forme parte del conjunto junto con otra relación entre las mismas dos variables o dos completamente distintas. Por ejemplo:

x>y AND y>z. La computadora resolverá que el par (x=4,y=3) y (y=4,z=1) si es parte de la intersección de los conjuntos y por ende el condicional resultará verdadero. Las variables x,y y z normalmente serán producto de cálculos dentro de un programa. Por su parte, la opción (x=4,y=4) independientemente de lo que valga z, no formará parte de esta intersección y por ende, la condición resultará falsa.

x>y OR y>z. La computadora resolverá que el par (x=4,y=3) y (y=4,z=1) si es parte de la intersección de los conjuntos y por ende el condicional resultará verdadero. Las variables x,y y z normalmente serán producto de cálculos dentro de un programa. Por su parte, la opción (x=4,y=4), siempre y cuando exista una z<y será verdadera (ejemplo, z=-10). En este caso el OR evalúa si las relaciones forman parte de la Unión de los dos conjuntos.

Para poder evaluar fácilmente estas condiciones y su pertenencia a un “verdadero” o “falso” se usa la herramienta de tablas de verdad:

X > Y	Y > Z	AND	OR
0	0	0	0
1	0	0	1

0	1	0	1
1	1	1	1

En la tabla anterior se puede analizar: cuando existan tres números X, Y, Z con relaciones marcadas por el mayor que, y alguna de las dos sea verdadera, el operador OR resolverá la condición como verdadera. Por su cuenta, el operador AND solo podrá resolver “verdadero” cuando AMBAS relaciones sean ciertas.

Ejercicio: elija valores X,Y,Z para los cuales tanto AND como OR resuelvan verdadero.

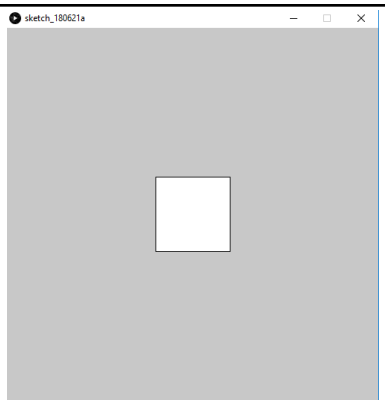
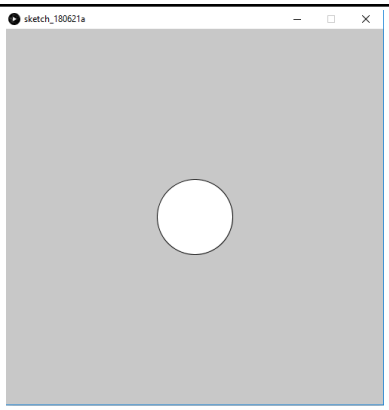
Basado en las matemáticas discretas, las condiciones determinan estados de “verdadero” o “falso” y por ende la ejecución de instrucciones. El esqueleto para los condicionales es:

if(condition){
// code to execute on condition = true
}else{
// code to execute on condition = false
}
	if(condition1){
	// code to execute on condition = true
	}else if(condition2){
	// code to execute on condition2 = true
	}else{
	// code to execute on condition1 and conditio2 = false
	}

Este formato puede ir en cualquier lugar donde se pueda colocar código de actuación como pintar. El siguiente es un ejemplo basado en una entrada de usuario.

```
import javax.swing.JOptionPane;

size(500,500);

String response = JOptionPane.showInputDialog(null, "What is your name");

if( response.equals("joe") ){

  rect( 200,200,100,100);

}else{
  ellipse( 250,250,100,100);
}
```

Cuando se ejecuta, dependiendo de la entrada del usuario, podrá hacer una de estas dos opciones:

	
Cuando el usuario ingresa la palabra “joe” en el cuadro de diálogo	Cuando el usuario ingresa cualquier otro carácter en el cuadro de diálogo

Ejercicio: del programa de poción del mago en la unidad pasada, decide, si el nombre del mago es “Mandrake” la poción debe dibujarse en x=50, mientras que si el mago se llama “Merlin” se deberá dibujar en x=300, de cualquier otra manera se deberá dibujar en el centro.

Funciones

Normalmente, un programa de manera profesional incluirá una cantidad importante de líneas de código. Una ventaja de las computadoras es la capacidad de poder repetir rutinas y código sin necesidad de una generación explicita por parte del programador. Durante esta unidad se introducirá el concepto de funciones como agrupaciones de líneas de código que podrán ser ejecutadas de manera repetida para poder “ahorrar” al programador, el escribir 10 o 15 veces las mismas secuencias.

Dentro del siguiente ejemplo se hacen “cruces” a partir del uso de 2 “rect”. Una opción para poder hacer 5 cruces es:

```
int px = 200;
int py = 200;


size(500,500);

px = 200;
py = 200;
rect(px-60,py-20,120,40);
rect(px-20,py-60,40,120);


px = 300;
py = 300;
rect(px-60,py-20,120,40);
rect(px-20,py-60,40,120);


px = 400;
py = 400;
rect(px-60,py-20,120,40);
rect(px-20,py-60,40,120);


px = 300;
py = 200;
rect(px-60,py-20,120,40);
rect(px-20,py-60,40,120);


px = 150;
py = 400;
rect(px-60,py-20,120,40);
rect(px-20,py-60,40,120);
```

Como se puede analizar, las líneas son un tanto repetitivas y lo único que varía es el valor de **px** y **py** a lo largo del programa. Justo una de las primeras ayudas que propone el uso de un lenguaje, es el agrupamiento de estas líneas y la generación de un “alias” con el cual poder “mandarlo llamar”. En esencia, lo que se busca es lo siguiente:

1. Asignar una palabra especial para la agrupación de líneas de código
2. Asignar las líneas de código a esa palabra
3. Sustituir las líneas de código que “hacían algo” por esa palabra
4. Cada vez que la computadora lea esa palabra, “expandirá” las líneas de código para ejecutarlas como si estuvieran ahí.

De esta forma el código anterior podría ser transformado en el siguiente:

```
int px = 200;
int py = 200;

void drawCross(){
  rect(px-60,py-20,120,40);
  rect(px-20,py-60,40,120);
}

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

  px = 200;
  py = 200;
  drawCross();

  px = 300;
  py = 300;
  drawCross();

  px = 400;
  py = 400;
  drawCross();

  px = 300;
  py = 200;
  drawCross();

  px = 150;
  py = 400;
  drawCross();
}
```

Nota: al momento de necesitar "funciones" para poder agrupar código el ambiente de Processing requiere que el código principal se encuentre dentro de "otra función" llamada setup() o draw(). Esto es para que pueda mantener por separado las líneas de código y pueda saber en donde empezar. La función "setup" está predefinida en Processing y es una especie de "código" principal o inicial. Más adelante se detallará el por qué se ejecuta.

Aunque el código, parece haber sido incrementado, en realidad se ha ordenado y es un poco más comprensible. Se ha generado una rutina que ahora en el nombre describe lo que hacen las líneas de código y además simplifica la primer lectura.

Ejercicio: Del ejercicio donde se realizó una "cara" en la primera unidad, colocar una función que ahora la englobe y se puedan pintar dos o tres.

Sin embargo, el poder total de las funciones no se visualiza hasta que se utilizan los "PARÁMETROS". Un parámetro es la "materia prima" que necesitan las funciones para que el código resuelva las operaciones. Un modelo básico de función entendiendo los parámetros es el de una "máquina". Las funciones son máquinas simples que toman materia prima y ejecutan una serie de instrucciones, incluso, como se verá más adelante, podrán "entregar" un producto terminado relativo a esta máquina para poder conectarse con otras funciones (máquinas).

Analizando el código, se puede hacer la siguiente adecuación:

```
void drawCross(int px, int py){
  rect(px-60,py-20,120,40);
  rect(px-20,py-60,40,120);
}

void setup(){
  size(500,500);
  drawCross(200,200);
  drawCross(300,300);
  drawCross(400,400);
  drawCross(300,200);
  drawCross(150,400);
}
```

Ahora es notable la optimización del código. En lugar de colocar 4 líneas para hacer la cruz, ahora se expanden las líneas cuando la computadora las ejecuta y los valores de **px** y **py** se han colocado como parámetros.

Es importante entender la generación de las funciones en dos pasos:

1. La creación de la función
2. La ejecución de la función

Dentro de la creación de la función se debe pensar en una especie de “subprograma” en el que por un momento, solo se debe diseñar un segmento de la solución orientado a que los “parámetros” son las grandes variables que se necesitan (análogo a pedir las variables al usuario vía ShowMessageDialog), y definiendo el resto del código con respecto a ellas.

Una vez que ya se ha creado la función, cuando se requiere “ejecutar”, dentro de la línea principal de programa, se coloca la palabra con la que se englobaron las líneas de código, seguido de paréntesis en los que en el orden en que se definieron, se colocarán los valores de las variables. Justo cuando se ejecute, las variables previamente definidas en la función, tomarán un valor y ahora ejecutarán las líneas englobadas con esos valores. Las variables definidas sin tener valor cuando se crea la función se llamarán argumentos formales, mientras que los valores “pasados” a la función dentro de los paréntesis desde la línea principal de ejecución se llamarán argumentos actuales (esto es un posible barbarismo a raíz del vocablo inglés “actual” que se puede traducir como el “final”).

De hecho, se han estado usando funciones definidas dentro de Processing como rect(x,y,w,h) que significa que los 4 parámetros los utilizará como la posición en pantalla **x**, y **y** el tamaño en ancho y en alto del rectángulo.

Una vez que han sido entendidas las funciones como un integrador de contenido o la generación de un bloque de programación, es importante entender su verdadero significado, que es la de producir “maquinarias” completas que son capaces de integrar materia prima y producir información útil y amable para el resto del programa.

Funciones con retorno

Las funciones con retorno son exactamente las mismas funciones vistas anteriormente pero incluyen el uso de una última palabra llamada “return”. La analogía para entenderlas es simple:

Una función tiene el objetivo de realizar una tarea, tanto como un procesador de alimentos puede destruir la forma más “compleja” de zanahorias o manzanas, y las “transforma” en algo más interesante para la tarea que viene. Las funciones “procesan” las variables e incluso usan otras funciones internamente para integrar un resultado que es “fácilmente” visible para quien está “usando la función” (refiriéndose al programar que la está usando).

El siguiente ejemplo es suficientemente complejo para identificar los usos de las funciones. El programa utilizará funciones para pintar una cuadrícula, posteriormente pedirá al usuario que ingrese la cuadra en donde está y a la que quiere ir. Posteriormente el programa responderá la cantidad en distancia lineal (si el personaje pudiera volar sobre los edificios) y la distancia en cuadras.

Normalmente hay muchas funciones que ya vienen implementadas dentro de un lenguaje, normalmente es te es una de las razones para poder elegirlo como plataforma de desarrollo. La otra es la facilidad con la que vemos que las instrucciones son puestas dentro de la computadora, Un programa de “alto nivel” requerirá pocas lineas para generar tareas complejas, mientras que un programa de bajo nivel, requerirá muchas línea para una tarea de la misma complejidad que el primero. La ventaja del segundo radica en el control y optimización que se puede tener del proceso, sin embargo requiere un alto reconocimiento del mismo.

Para el ejemplo se utilizarán funciones que ya existen dentro de processing y se crearán algunas que no existen para lograr que el procedimiento final se “vea”/”lea” fácil y lógico, casi como si escribiéramos un algoritmo muy general.

El algoritmo que se seguirá es el siguiente:

1. Pintar el plano de la ciudad identificando 5 filas de 4 cuadras cada una
2. Preguntar al usuario la cuadra en donde está, separando la fila y la columna (x1 y y1).
3. Preguntar al usuario la cuadra a dónde desea ir, separando la fila y la columna (x2 y y2).
4. Para temas de simplificar los cálculos, cada cuadra medirá 50 metros. Las calles intermedias serán despreciables.
5. Responder al usuario a partir del uso de la función text (sin retorno).

El método para resolver será el siguiente:

1. Generar una función que pinte 4 cuadras
2. Generar una función que utilice la anterior para pintar 5 veces

3. Generar una función que pida toda la información
4. Generar una función que calcule la distancia lineal
5. Generar una función que calcule la distancia en cuadras
6. Usar las funciones dentro de la función setup

Resolución:

```
import javax.swing.JOptionPane;
import java.lang.Math.*;



int blockX1;
int blockX2;
int blockY1;
int blockY2;


void setup(){

  size(800,600);

  drawLineBlocks(50,50);
  delay(100);
  getInformation();
  float d1 = getLinearDistance(blockX1, blockY1, blockX2, blockY2);
  int d2 = getBlocksDistance(blockX1, blockY1, blockX2, blockY2);

  text("Distancia lineal es " + d1 , 40,550);

  text("Distancia en cuadras es " + d2 , 240,550);
}
```

```
void drawBlocks(int x, int y){

    fill(0);
    rect(x,y, 20,20);
    fill(255);
    text("1",x+10,y+10);
    x= x+25;
    fill(0);
    rect(x,y, 20,20);
    fill(255);
    text("2",x+10,y+10);
    x= x+25;
    fill(0);
    rect(x,y, 20,20);
    fill(255);
    text("3",x+10,y+10);
    x= x+25;
    fill(0);
    rect(x,y, 20,20);
    fill(255);
    text("4",x+10,y+10);
    x= x+25;
    fill(0);
    rect(x,y, 20,20);
    fill(255);
    text("5",x+10,y+10);

}

void drawLineBlocks(int sx, int sy){
  drawBlocks(sx,sy);
  text("1",sx-20,sy);
  sy+=25;
  drawBlocks(sx,75);
  text("2",sx-20,sy);
  sy+=25;
  drawBlocks(sx,100);
  text("3",sx-20,sy);
  sy+=25;
  drawBlocks(sx,125);
  text("4",sx-20,sy);
  sy+=25;
  drawBlocks(sx,150);
  text("5",sx-20,sy);
}
```

```
void getInformation(){
  blockX1 = Integer.parseInt( JOptionPane.showInputDialog("Columna 1") );
  blockY1 = Integer.parseInt( JOptionPane.showInputDialog("Fila 1") );
  blockX2 = Integer.parseInt( JOptionPane.showInputDialog("Columna 2") );
  blockY2 = Integer.parseInt( JOptionPane.showInputDialog("Fila 2") );
}

float getLinearDistance(int x1, int y1, int x2, int y2){

  // uses distance between two points formula

  int dX = x2-x1;
  int dY = y2-y1;

  float dist = sqrt(dX*dX + dY*dY);

  return dist*50.0f;
}

int getBlocksDistance(int x1, int y1, int x2, int y2){

  int dX = abs(x2-x1);
  int dY = abs(y2-y1);


  return (dX+dY);
}
```

Particularmente la función setup es importante por que prácticamente es una traducción del algoritmo. El resto de las funciones son la manera en que el lenguaje puede “compartamentalizar” o agrupar contenido e incluso generar una respuesta (parecida a un mini programa dentro del gran programa). Dentro de este ejemplo se analizó la distancia entre dos puntos. Esta fórmula tiene su base en un triángulo rectángulo, ¿sabes por qué?

Ejercicio: pinta las cuadras que el usuario seleccionó despues de haber calculado la distancia.

Dudas:

Ejercicio: elija valores X,Y,Z para los cuales tanto AND como OR resuelvan verdadero. -->Aquí, AND y OR, según yo había investigado sobre eso pero no lo recuerdo muy bien para poder hacer el ejercicio.

AND y OR. TIene que ver con la siguiente tabla de verdad>

AND

Entrada 1	Entrada 2	Salidas
F	F	F
F	V	F
V	F	F
V	V	V

OR

Entrada 1	Entrada 2	Salidas
F	F	F
F	V	V
V	F	V
V	V	V

Esto significa que para dos condiciones por ejemplo if(a>5 && b<4) a>5 representa la entrada 1 y b<4 representa la entrada 2. Si a ya en el programa vale a = 3 y b=6, se evalúan por parte de la computadora los valores:

if(3>5 && 6<4) dado que ambos elementos son falsos, y es un AND, se revisa la tabla de verdad y ya que la salida es Falsa, la computadora reduce a :

if(F && F) que la salida de F && F es F :=> if(F) y los If, no pueden ejecutar las instrucciones que vienen por delante a menos que sean if(V) entonces no entra al codigo del if.

Como ves si?

Excelente!! Ahora entiendo el problema.

Las funciones, aún no he tenido la oportunidad de hacerlo, tuve que llevar mi laptop por mantenimiento pero por lo que he leído han surgido unas dudas:

-->2.Generar una función que utilice la anterior para pintar 5 veces, ¿cómo pintar 5 veces?

Claro, hay que utilizar la función generada en (1) dentro de la función 2 para que pinte lineas de 4 cuadros. COmo ves, en realidad tu solo hiciste una funcion que hacía algo y la reusas para poder ahorrarte tiempo.

Oh, ya, entiendo, no sabía a que debía pintar las 5 veces, jeje.

Setup,

Es que el método es completo:

1. Generar una función que pinte 4 cuadras
2. Generar una función que utilice la anterior para pintar 5 veces
3. Generar una función que pida toda la información
4. Generar una función que calcule la distancia lineal
5. Generar una función que calcule la distancia en cuadras
6. Usar las funciones dentro de la función setup

Por eso la resolución.

Sí, por favor, tú me dices cuándo pueda responder. :3

La función de Setup, en qué sería bueno para utilizar “Setup”? El setup es la funcion de entrada para el resto del programa. Solo funciona en processing, y es la preparacion, como en Unity el START.

¡Ah! Con razón, lo había intentando en otro editor y no resultaba, jeje. Muy bien, la siguiente duda es hay dos tipos de dato: float e int, significa que también debo usar float como 25.3? E Int como número entero, cierto?Exacto!!! Muy bien. EN realidad hay dos tipos de decimales y dos tipos de entero, es decir hay: double y float, y long e int. Pero los mas usados son float e int asi como tu lo dijiste.

Muy bien, y es cierto, había leído sobre double y long pero como dices que es más común usar float e int, entendido.

Por cierto, tus ejercicios son entretenidos y creativos, me encanta. :3 Creo que por el momento esas son mis dudas. Excelente. Te muestro un ejemplo de ciclo y lo explico y con eso acabamos para darte chance de hacer lo de funciones. Excelente!

Excelente.

Escribe todas las dudas, en lo que checas las funciones yo escribo la primera parte de los ciclos.

Le sigo? Seguirian los ciclos.

Ciclos

Una de las principales labores de la programación es la posibilidad de repetir las operaciones. De hecho la primera aplicación real sobre negocios que se generó dentro de la industria, fue el cálculo de nómina. En suma, los ciclos representan muchas de las capacidades del ser humano para procesar ciertos datos en cantidades importantes como lo serían las integrales, series o sumatorias. Cuando se agregan estos elementos, se multiplican los beneficios de estas herramientas pudiendo procesar incluso otro tipo de métodos como FEM (análisis de elemento finito).

Los ciclos, en su funcionamiento cuentan con dos tipos principales de variables (en realidad son variables normales, sin embargo, los programadores las han clasificado por su uso principal), los contadores y los acumuladores. Como se verá más adelante tanto los contadores como los acumuladores hacen lo mismo, van “agregando” una cantidad a la variable, sin embargo están usados de maneras distintas.

El ciclo más básico que se puede implementar es:

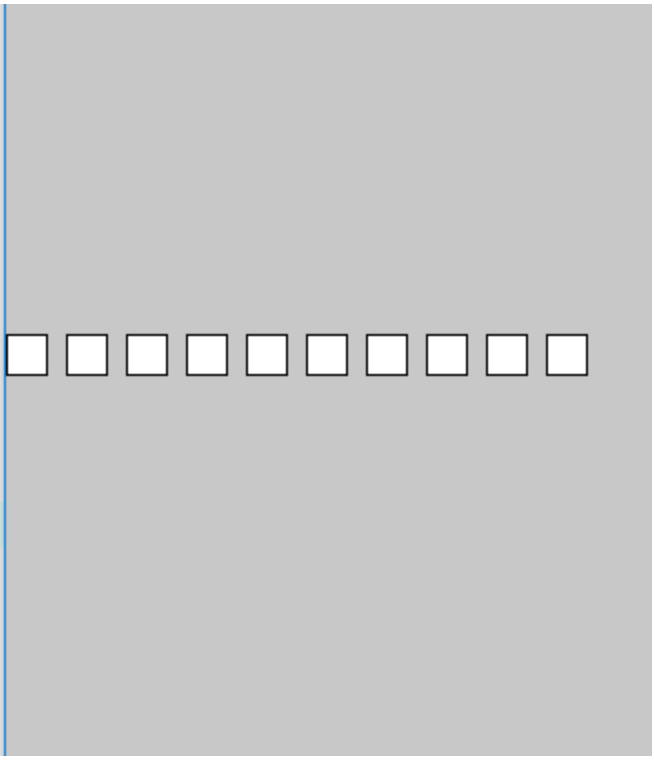
```
while ( condition ) {

// do something

}
```

Este ciclo SIEMPRE deberá ir dentro de una función para que pueda ser ejecutado (en este sentido es parecido al condicional IF).

Cabe resaltar que la condición del “while” sigue las mismas reglas que para cualquier condición de IF pudiendo colocar ANDs y ORs. El objetivo es : mientras la condición es verdadera, el cuerpo “do something” del ciclo se va a ejecutar. En el momento que la condición se vuelva falsa, el programa “saltará” el cuerpo del while y continuará el código por debajo.

En seguida se analiza un programa básico :

```
void setup(){

    size(600,600);

}

void draw(){

  int i =0 ;
   while( i < 10){
     rect(i*30,200,20,20);
     i++;
   }


}
```



Como en todos los programas, se empieza con la función setup() para ejecutar la línea de size() que inicializa la ventana y ,a coloca de un tamaño igual a 600px por 600px.

Una vez pasando eso, en cada “frame” de ejecución, la computadora inicializa la variable i a 0.

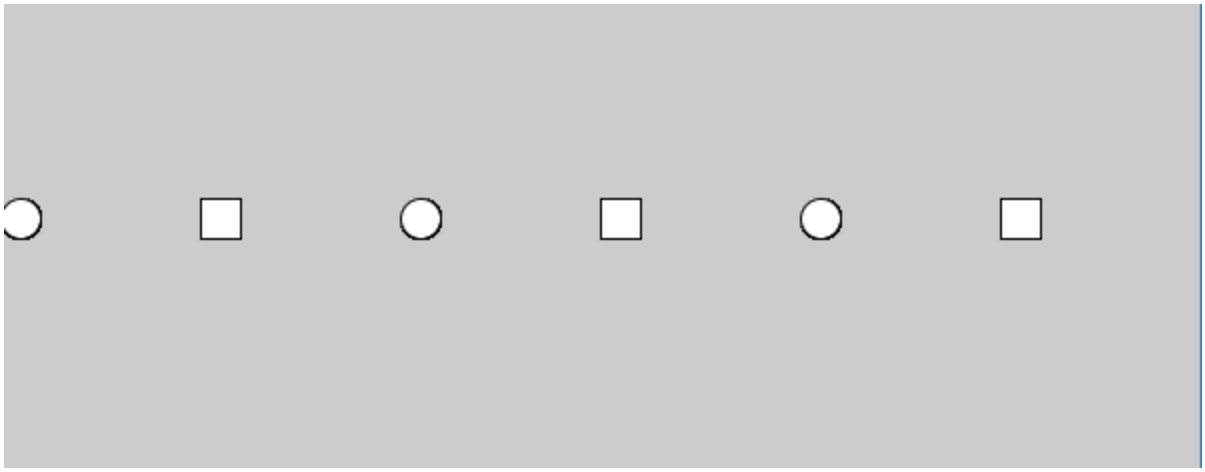
Durante 10 veces, se pintan (en el mismo frame) un rectángulo pero, en cada iteración, la X del rect, en lugar de quedarse en un solo lugar, se cambia con respecto a “i”. Al final, la “i”, se incrementa de uno en uno, y eventualmente hace que la condición “i<10” sea falsa. El mismo programa pero sin ciclos se generaría de la siguiente manera:

```
void draw(){

    rect(0*30,200,20,20);
    rect(1*30,200,20,20);
    rect(2*30,200,20,20);
    rect(3*30,200,20,20);
    rect(4*30,200,20,20);
    rect(5*30,200,20,20);
    rect(6*30,200,20,20);
    rect(7*30,200,20,20);
    rect(8*30,200,20,20);
    rect(9*30,200,20,20);

}
```

Es claro que en menos líneas se logra un programa que hace lo mismo, en suma es mucho más fácil variar la funcionalidad gracias a que es menos código que reemplazar. Cabe destacar que el ciclo empieza con i=0 y termina cuando i=10 dado que la condición es i<10.

Ejercicio:

1. Genera un programa que coloque 1 círculo cada 50 pixeles en una pantalla de 300 pixeles de largo. No lo hagas repitiendo la linea, sino utilizando ciclos.
2. Genera un programa que coloque 1 ciculo cada 100 pixeles y un rectángulo cada 100 pixeles de la siguiente manera (clave, usa condiciones):



Puedes usar dos ciclos o 1.

1.