|  |  |  |
| --- | --- | --- |
| Texto  Descripción generada automáticamente con confianza media | TECNICATURA UNIVERSITARIA EN DISEÑO INTEGRAL DE VIDEOJUEGOS  FACULTAD DE INGENIERÍA  Universidad Nacional de Jujuy |  |

*Profesores:*

*Mg. Ing. Ariel Alejandro Vega*

*Ing. Carolina Cecilia Apaza*

*Año*

Trabajo Práctico

N°1 – parte dos

Muñoz Romina Alejandra

LU: TUV000688

**FUNDAMENTOS DE PROGRAMACIÓN ORIENTADA A OBJETOS**

**ÍNDICE**

[Sección Análisis – Diseño y Codificación de algoritmos – Aplicación de estructuras de control 3](#_Toc164003896)

[**Ejercicio 12:** 3](#_Toc164003897)

[**Ejercicio 13:** 3](#_Toc164003898)

[**Ejercicio 14:** 4](#_Toc164003899)

[**Ejercicio 15:** 6](#_Toc164003900)

[**Ejercicio 16:** 7](#_Toc164003901)

[**Ejercicio 17:** 8](#_Toc164003902)

[**Ejercicio 18:** 10](#_Toc164003903)

[**Ejercicio 19:** 11](#_Toc164003904)

[**Ejercicio 20:** 13](#_Toc164003905)

[**Ejercicio 21:** 14](#_Toc164003906)

[**Ejercicio 22:** 16](#_Toc164003907)

[Conclusión 18](#_Toc164003908)

[Fuentes bibliográficas 18](#_Toc164003909)

# **Sección Análisis – Diseño y Codificación de algoritmos – Aplicación de estructuras de control**

## **Ejercicio 12:**

Un problema sencillo. Deberá pedir por teclado al usuario un nombre y posteriormente realizará la presentación en pantalla de un saludo con el nombre indicado.

**Fase de Análisis:**

• Especificación del Problema: Mostrar un saludo en la pantalla con un nombre indicado por el usuario

• Análisis:

Datos de Entrada: nombreUsuario: String

Datos de Salida: saludo: String

Proceso:

Guardar el saludo // pedir al usuario que ingrese un nombre // guardar ese nombre // juntar el saludo con el nombre y mostrarlo en la pantalla.

**Fase de Diseño:**

|  |
| --- |
| **ENTIDAD QUE RESUELVE EL PROBLEMA:** Usuario |
| **VARIABLES:**  nombreUsuario: String  saludo: String |
| **NOMBRE ALGORITMO:** Enviar\_saludo  **PROCESO DEL ALGORITMO:**   1. *Leer* nombreUsuario 2. saludo 🡨 “Hola” + nombreUsuario 3. *Mostrar* saludo   **FIN ALGORITMO** |

## **Ejercicio 13:**

Será común resolver problemas utilizando variables. Calcule el perímetro y área de un rectángulo dada su base y su altura.

**Fase de Análisis:**

• Especificación del Problema: Calcular el perímetro y área de un rectángulo según su base y altura

• Análisis:

Datos de Entrada: base: Real

altura: Real

Datos de Salida: perímetro: Real

área: Real

Proceso:

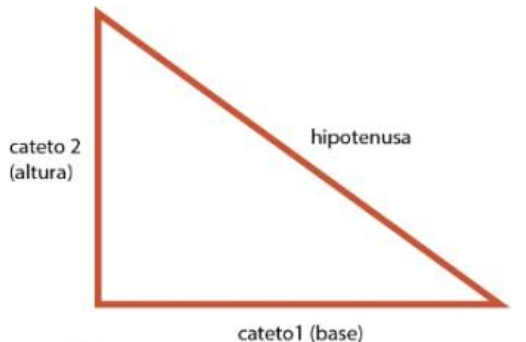
Cargar los datos de la base y la altura del rectángulo // calcular el perímetro usando la fórmula 2\*(base + altura) // calcular el área usando la formula base \* altura // mostrar el perímetro y el área ya calculada.

**Fase de Diseño:**

|  |
| --- |
| **ENTIDAD QUE RESUELVE EL PROBLEMA:** calculadora |
| **VARIABLES:**  base: Real  altura: Real  perímetro: Real  área: Real |
| **NOMBRE ALGORITMO:** Calcular\_área\_perímetro  **PROCESO DEL ALGORITMO:**   1. *Leer* base 2. *Leer* altura 3. perímetro 🡨 2\*(base + altura) 4. área 🡨 base \* altura 5. *Mostrar* perímetro 6. *Mostrar* área   **FIN ALGORITMO** |

## **Ejercicio 14:**

Una ayuda importante al momento de resolver problemas con algoritmos es asumir que su gran amigo son las matemáticas. Obtenga la hipotenusa de un triángulo rectángulo conociendo sus catetos.



**Fase de Análisis:**

• Especificación del Problema: Calcular la hipotenusa de un triángulo rectángulo con sus catetos.

• Análisis:

Datos de Entrada: cateto1: Real

cateto2: Real

Datos de Salida: hipotenusa: Real

Proceso:

Cargar los datos del cateto1 y del cateto2 del triángulo rectángulo // calcular la hipotenusa usando la fórmula ((cateto1 ^ 2) + (cateto2 ^ 2)) ^ (1 / 2) // mostrar la hipotenusa del triángulo rectángulo ya calculado.

**Fase de Diseño:**

|  |
| --- |
| **ENTIDAD QUE RESUELVE EL PROBLEMA:** calculadora |
| **VARIABLES:**  cateto1: Real  cateto2: Real  hipotenusa: Real |
| **NOMBRE ALGORITMO:** Calcular\_hipotenusa  **PROCESO DEL ALGORITMO:**   1. *Leer* cateto1 2. *Leer* cateto2 3. hipotenusa 🡨 ((cateto1 ^ 2) + (cateto2 ^ 2)) ^ (1 / 2) 4. *Mostrar* hipotenusa   **FIN ALGORITMO** |

## **Ejercicio 15:**

Si viste algo de los apuntes y vídeos, esto debería ser muy fácil de resolver. Dados dos números permita calcular la suma, resta, multiplicación y división de estos. Considere que cada una de estas operaciones es un algoritmo cuando realice el diseño. Obviamente muestre los resultados.

**Fase de Análisis:**

• Especificación del Problema: Calcular la suma, resta, multiplicación y división de dos números en algoritmos separados.

• Análisis:

Datos de Entrada: numero1: Real

numero2: Real

Datos de Salida: resultadoSum: Real

resultadoRest: Real

resultadoMult: Real

resultadoDiv: Real

Proceso:

Ingresar el valor de los dos números // crear el 1er algoritmo // sumar los dos números // mostrar el resultado de la suma en la pantalla // crear el 2do algoritmo // restar los dos números // mostrar el resultado de la resta en la pantalla // crear el 3er algoritmo // multiplicar los dos números // mostrar el resultado de la multiplicación en la pantalla // crear el 4to y último algoritmo // dividir los dos números // mostrar el resultado de la división en la pantalla

**Fase de Diseño:**

|  |
| --- |
| **ENTIDAD QUE RESUELVE EL PROBLEMA:** Calculadora |
| **VARIABLES:**  numero1: Real  numero2: Real |
| **NOMBRE ALGORITMO1:** Suma  **PROCESO DEL ALGORITMO1:**   1. *Leer* numero1 2. *Leer* numero2 3. resultadoSum 🡨 numero1 + numero2 4. *Mostrar* resultadoSum   **FIN ALGORITMO1** |
| **NOMBRE ALGORITMO2:** Resta  **PROCESO DEL ALGORITMO2:**   1. *Leer* numero1 2. *Leer* numero2 3. resultadoRest 🡨 numero1 - numero2 4. *Mostrar* resultadoRest   **FIN ALGORITMO2** |
| **NOMBRE ALGORITMO3:** Multiplicación  **PROCESO DEL ALGORITMO3:**   1. *Leer* numero1 2. *Leer* numero2 3. resultadoMult 🡨 numero1 \* numero2 4. *Mostrar* resultadoMult   **FIN ALGORITMO3** |
| **NOMBRE ALGORITMO4:** División  **PROCESO DEL ALGORITMO4:**   1. *Leer* numero1 2. *Leer* numero2 3. resultadoDiv 🡨 numero1 / numero2 4. *Mostrar* resultadoDiv   **FIN ALGORITMO4** |

## **Ejercicio 16:**

Necesitamos convertir una temperatura Fahrenheit en grados Celsius. Si no conoce la forma en la que se realiza esta conversión, debería investigarlo; para eso sirve la etapa de análisis. Pero como somos buenos, daremos una ayuda:



**Fase de Análisis:**

• Especificación del Problema: Pasar una temperatura que se encuentra en Fahrenheit a grados Celsius.

• Análisis:

Datos de Entrada: tempFahrenheit: Real

Datos de Salida: tempCelsius: Real

Proceso:

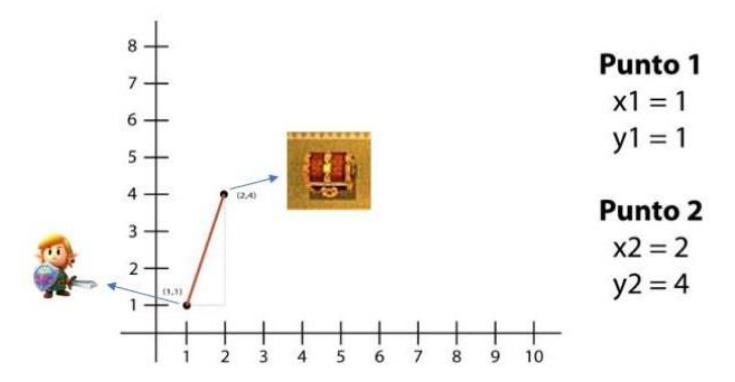
Cargar los datos de la temperatura en Fahrenheit // convertir esa temperatura en grados Celsius usando la fórmula tempCelsius= (tempFahrenheit – 32) / 1.8 // mostrar la temperatura en grados Celsius.

**Fase de Diseño:**

|  |
| --- |
| **ENTIDAD QUE RESUELVE EL PROBLEMA:** calculadora |
| **VARIABLES:**  tempFahrenheit: Real  tempCelsius: Real |
| **NOMBRE ALGORITMO:** Pasar\_Fahrenheit\_a\_Celsius  **PROCESO DEL ALGORITMO:**   1. *Leer* tempFahrenheit 2. tempCelsius 🡨 (tempFahrenheit – 32) / 1.8 3. *Mostrar* tempCelsius   **FIN ALGORITMO** |

## **Ejercicio 17:**

Si queremos representar personajes o power ups (premios) en la pantalla debemos primero ubicarlos en alguna posición dentro de la pantalla. Imagine que está en un juego donde un power up desaparece porque el personaje se acerca a una distancia de x unidades, sin importar por donde se acerque. Por tanto, para que desaparezca, en primer lugar, hay que determinar esa distancia. La forma de representar la posición de un objeto en la pantalla es a través de las coordenadas de un punto. Suponga que la posición de Link está representada por la coordenada (𝑥1, 𝑦1), mientras que las de la caja de tesoro se halla en la posición (𝑥2, 𝑦2). Si observa con detenimiento se observa la conformación de un triángulo rectángulo, por lo que es posible aplicar Pitágoras para obtener la distancia.



Para esto debe calcular el tamaño de los catetos y luego aplicar el teorema. Halle la distancia entre ambos objetos. Cuando programe, represente a Link con un Circulo, y al tesoro con un cuadrado. Además, mueva a Link mediante el mouse.

**Fase de Análisis:**

• Especificación del Problema: Calcular el tamaño de los catetos para poder aplicar teorema de Pitágoras y hallar la distancia entre Link y el tesoro.

• Análisis:

Datos de Entrada: x1: 1 (Real)

y1: 1 (Real)

x2: 2 (Real)

y2: 4 (Real)

Datos de Salida: distLinkTes: Real

Proceso:

Cargar las coordenadas x1, y1, x2 e y2 // Calcular el tamaño de los catetos aplicando las fórmulas x2 – x1 y y1 – y2 // Con el resultado de las restas aplicar teorema usando la formula ((cateto1 ^ 2) + (cateto2 ^ 2)) ^ (1 / 2) para calcular la distancia entre Link y el tesoro // Mostrar la distancia calculada entre Link y el tesoro.

**Fase de Diseño:**

|  |
| --- |
| **ENTIDAD QUE RESUELVE EL PROBLEMA:** Usuario |
| **VARIABLES:**  x1: Real  y1: Real  x2: Real  y2: Real |
| **NOMBRE ALGORITMO:** Distancia\_Link\_tesoro  **PROCESO DEL ALGORITMO:**   1. *Leer* x1= 1 2. *Leer* y1= 1 3. *Leer* x2= 2 4. *Leer* y2= 4 5. cateto1 🡨 x1 – x2 6. cateto2 🡨 y1 – y2 7. *Mostrar* cateto1= 1 8. *Mostrar* cateto2= 3 9. distLinkTes 🡨 ((cateto1 ^ 2) + (cateto2 ^ 2)) ^ (1 / 2) 10. *Mostrar* distLinkTes= 3.16227…   **FIN ALGORITMO** |

## **Ejercicio 18:**

Desarrolle el análisis y diseño de un algoritmo que permita obtener las raíces de una ecuación de segundo grado. Además, utilice la estructura según para el análisis de la discriminante de la ecuación cuadrática. Obviamente codifique en Processing.

**Fase de Análisis:**

• Especificación del Problema: Obtener las raíces y la discriminante de una ecuación cuadrática.

• Análisis:

Datos de Entrada: a: Real

b: Real

c: Real

Datos de Salida: x1: Real

x2: Real

disc: Real

Proceso:

Una ecuación cuadrática es igual a **“ax2 + bx + c”**, por lo tanto:

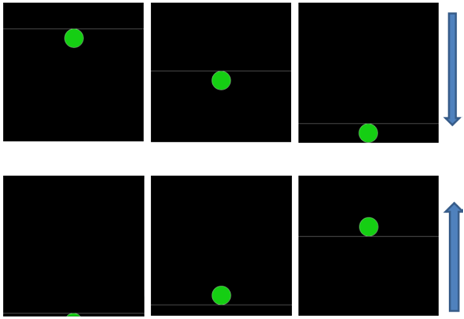
Cargar las variables a, b y c // calcular la raíz x1 usando la fórmula (– b + (((b ^ 2) – 4 \* a \* c)) ^ (1 / 2)) / (2 \* a) // calcular la raíz x2 usando la fórmula (– b – (((b ^ 2) – 4 \* a \* c)) ^ (1 / 2)) / (2 \* a) // mostrar las raíces x1 y x2 // calcular la discriminante usando la formula (b ^ 2) – 4 \* a \* c // mostrar la discriminante.

**Fase de Diseño:**

|  |
| --- |
| **ENTIDAD QUE RESUELVE EL PROBLEMA:** usuario |
| **VARIABLES:**  a: Real  b: Real  c: Real |
| **NOMBRE ALGORITMO:** Calcular\_raíces\_discriminante  **PROCESO DEL ALGORITMO:**   1. *Leer* a 2. *Leer* b 3. *Leer* c 4. x1 🡨 ((– b + ((b ^ 2) – 4 \* a \* c)) ^ (1 / 2)) / (2 \* a) 5. x2 🡨 ((– b – ((b ^ 2) – 4 \* a \* c)) ^ (1 / 2)) / (2 \* a) 6. *Mostrar* x1 7. *Mostrar* x2 8. disc 🡨 (b ^ 2) – 4 \* a \* c 9. *Mostrar* disc   **FIN ALGORITMO** |

## **Ejercicio 19:**

Declare las variables necesarias para dibujar una línea que se dibuja desde las coordenadas iniciales del lienzo y se extiende por todo el ancho. Sobre el punto medio de la línea y a una distancia de 40px (en sentido vertical desde la línea) dibuje una elipse que tenga como ancho 80px y de alto 80px. Dentro de la función draw (), actualice las variables necesarias para que la línea desde su inicio se mueva en dirección hacia abajo arrastrando la elipse. Mantenga en cero el valor para background (). Cuando la línea supere la posición de la altura del lienzo, debe invertir su sentido, es decir dirigirse hacia arriba arrastrando la elipse. Cuando la línea alcance nuevamente el valor 0 para su posición en y, el desplazamiento debe ser hacia abajo y así sucesivamente.



**Fase de Análisis:**

• Especificación del Problema: Dibujar sobre el lienzo una línea que abarque todo el ancho de la pantalla desde el inicio del lienzo y un círculo de 80x80 que se encuentre justo en su punto medio y desplazado verticalmente por 40, después hacer que la línea se desplace hacia abajo arrastrando al círculo y cuando la línea supere el alto de lienzo, hacer que regrese arriba con el circulo, por último, hacer que este proceso se repita de manera infinita.

• Análisis:

Datos de Entrada: y1: coordenada

y2: coordenada

vy: entero

Datos de Salida: el circulo y la línea desplazándose de arriba debajo de manera infinita.

Proceso:

Crear el lienzo.

Cargar las variables y1, y2 y vy.

Crear la línea con sus respectivas coordenadas.

Crear el circulo con sus respectivas coordenadas y a su coordenada y2 sumarla con y1 para que esta coordenada se vuelva dependiente a y1 en el momento de ejecutarlo.

Marcar que y1 es igual a y1 + vy para que vaya incrementando su valor y así el circulo y la línea vayan descendiendo.

Usar la estructura selectiva Si para poder crear el bucle. Como condición en Si, aplicar el operador lógico O y poner: “*Si* y1 >= altoLienzo O y1 <= 0 *entonces*…”.

Después del entonces colocar que vy = – vy, al hacerlo el valor de vy será negativo y cuando y1 llegue al final del alto del lienzo este en vez de sumar le va a restar a y1 haciendo que la línea junto al círculo regrese a su posición original para después, cuando y1 vuelva a ser 0, volverse en un valor positivo repitiendo el ciclo de bajar y subir.

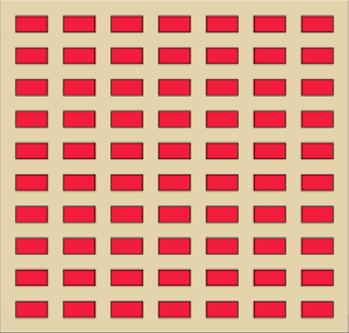
Después colocar que y2 = – y2, al hacerlo cuando y1 haya recorrido todo el alto del lienzo este se volverá negativo, regresara arriba sobre la línea y al y1 llegar a cero el circulo volverá a su posición original y se repetirá el ciclo otra vez.

**Fase de Diseño:**

|  |
| --- |
| **ENTIDAD QUE RESUELVE EL PROBLEMA:** Lienzo |
| **VARIABLES:**  y1: 0  y2: 40  vy: 3 |
| **NOMBRE ALGORITMO:** Dibujar\_línea\_círculo  **PROCESO DEL ALGORITMO:**   1. anchoLienzo 🡨 600 2. altoLienzo 🡨 600 3. *Leer* y1 4. *Leer* y2 5. *Leer* vy // velocidad de y 6. *Dibujar* línea donde un extremo sea en las coordenadas (0, y1) y el segundo en (anchoLienzo, y1) 7. *Dibujar* circulo en (anchoLienzo/2, y1+y2) con un tamaño de (80, 80) 8. y1 🡨 y1 + vy 9. *Si* y1 >= altoLienzo O y1 <= 0 *entonces* 10. y2 🡨 –y2 11. vy 🡨 –vy 12. *fin\_si*   **FIN ALGORITMO** |

## **Ejercicio 20:**

Dibuje en toda la extensión del lienzo de (440, 420) rectángulos de idénticas medidas (40 ancho y 20 de alto) y que mantengan una distancia de 20 pixeles entre ellos tanto horizontal como verticalmente. Utilice la estructura de control repetitiva for. El lienzo debería verse así:



**Fase de Análisis:**

• Especificación del Problema: Mostrar sobre un lienzo de 440 x 420 rectángulos de 40 x 20 a una distancia de 20 tanto horizontal como verticalmente aplicando el bucle for.

• Análisis:

Datos de Entrada: X, Y: coordenadas

ancho, alto, distancia: entero

Datos de Salida: los rectángulos ya dibujados

Proceso:

Cargar los datos de las coordenadas.

Cargar la distancia de los rectángulos.

Cargar el ancho y alto de los rectángulos.

Aplicar la estructura iterativa para, aplicando que x=distancia, en *hasta* anchoLienzo y en *con paso* (distancia+ancho).

Después de *hacer*,aplicar otra vez la estructura iterativa para, esta vez aplicando que y=distancia, en *hasta* altoLienzo y en *con paso* (distancia+alto).

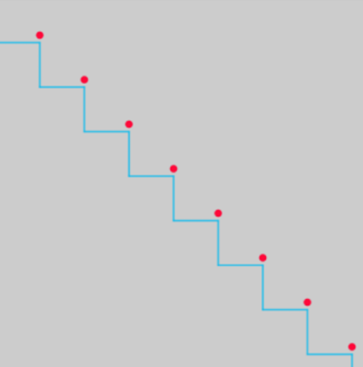
Dibujar rectángulo en (distancia, distancia) de (ancho, alto).

**Fase de Diseño:**

|  |
| --- |
| **ENTIDAD QUE RESUELVE EL PROBLEMA: lienzo** |
| **VARIABLES:**  X, Y: coordenadas  ancho, alto, distancia: entero |
| **NOMBRE ALGORITMO:** Dibujar\_rectángulos  **INICIO ALGORITMO:**   1. anchoLienzo 🡨 440 2. altoLienzo 🡨 420 3. distancia 🡨 20 4. ancho 🡨 40 5. alto 🡨 20 6. ***Para*** x🡨distancia ***hasta*** anchoLienzo ***con paso*** (distancia + ancho) ***hacer*** 7. ***Para*** y🡨distancia ***hasta*** altoLienzo ***con paso*** (distancia + alto) ***hacer*** 8. *Dibujar* rectángulo en (distancia, distancia) de (ancho, alto) 9. ***Fin\_para*** 10. ***Fin\_para***   **FIN ALGORITMO** |

## **Ejercicio 21:**

Utilizando la estructura de control repetitiva while() dibuje la siguiente imagen utilizando líneas que forman escalones y sobre cada borde de escalón se dibuje un punto de color rojo.



El tamaño del lienzo es size (500,500). La estructura while () se ejecuta dentro de la función setup (). La condición es que solo se dibuje dentro del lienzo. Utilice variables que puedan ayudar a la construcción del dibujo, por ej: x, y, anchoEscalon, altoEscalon, etc.

**Fase de Análisis:**

• Especificación del Problema: Mostrar sobre un lienzo de 500x500 líneas que forme un escalón con un punto en el borde de estos y usando la estructura while hacer que estos vayan descendiendo hasta formar una especie de escalera.

• Análisis:

Datos de Entrada: X, Y: coordenadas

ancho, alto: entero

Datos de Salida: la escalera más el punto en los bordes.

Proceso:

Cargar los datos de las coordenadas.

Cargar el ancho y alto de las líneas que formaran el escalón y que ubicara al punto.

Aplicar la estructura iterativa mientras, poniendo como condición que x <= width O y <= height.

Dibujar las líneas para que forme el escalón y el punto.

Hacer las sumas “x= x + ancho” e “y= y + alto” para que el valor de x e y vaya incrementando y así se termine formando la escalera con el punto arriba.

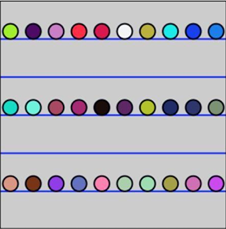
Finalizar función mientras.

**Fase de Diseño:**

|  |
| --- |
| **ENTIDAD QUE RESUELVE EL PROBLEMA: lienzo** |
| **VARIABLES:**  X, Y: coordenadas  ancho, alto, distancia: entero |
| **NOMBRE ALGORITMO:** Dibujar\_escalera+punto  **INICIO ALGORITMO:**   1. anchoLienzo 🡨 500 2. altoLienzo 🡨 500 3. ancho 🡨 60 4. alto 🡨 60 5. x 🡨 0 6. y 🡨 0 7. ***Mientras*** x <= anchoLienzo O y <= altoLienzo ***hacer*** 8. *Trazo* de color celeste y de 2 de grosor 9. *Dibujar* línea donde un extremo sea en x, (y+alto) y el otro en (x+ancho), (y+alto) 10. *Dibujar* línea donde un extremo sea en (x+ancho), (y+alto) y el otro en (x+ancho), (y+(alto\*2)) 11. *Trazo* de color rojo y de 9 de grosor 12. *Dibujar* punto en ((x + ancho), (y + (alto-10))) 13. x 🡨 x + ancho 14. y 🡨 y + alto 15. ***Fin\_mientras***   **FIN ALGORITMO** |

## **Ejercicio 22:**

Utilizando la estructura de control repetitiva do-while. Replique la siguiente imagen



La imagen debe ser construida desde la función setup (). Defina el tamaño del lienzo en size (600,600), verticalmente se divide el lienzo en franjas de igual medida, se deben dibujar los círculos sobre cada línea de por medio es decir en la línea 1 se dibujan círculos con distanciamiento, en la línea 2 no se dibuja y así sucesivamente. Las líneas tienen un color fijo, los círculos asumen colores aleatorios.

**Fase de Análisis:**

• Especificación del Problema: Usando la estructura do-while, mostrar sobre un lienzo de 600x600 líneas que dividan a este en 6 partes además de unos círculos que se repitan sobre las líneas cada dos de ellas con colores random.

• Análisis:

Datos de Entrada: x, y, x2, y2: coordenadas

anchoC, distancia: entero

Datos de Salida: la escalera más el punto en los bordes.

Proceso:

Cargar los datos de las coordenadas

Cargar el ancho del círculo.

Cargar la distancia que habrá entre ellos.

Aplicar la estructura iterativa hacer-mientras.

Dibujar las líneas que dividirán el lienzo.

Hacer la suma “y= y + (alto/6)” para que la línea se vaya repitiendo hasta dividir el lienzo.

Poner como condición “y <= alto” y finalizar función hacer-mientras.

Aplicar la estructura iterativa hacer-mientras.

Marcar que (x2 = distancia) para que la coordenada x2 dependa del valor de la distancia.

Aplicar nuevamente la estructura iterativa hacer-mientras.

Dibujar el circulo que se repetirá a lo ancho y alto del lienzo y hacer que tenga colores random.

Hacer la suma “x2= x2 + (distancia\*2)” para que x2 vaya incrementando y así el circulo se repita a lo ancho del lienzo.

Poner como condición “x2 <= ancho” y finalizar función hacer-mientras.

Realizar la suma “y2= y2 + (alto/3)” para que la hilera que se repite al inicio se repita de manera vertical cada 2 líneas.

Poner como condición y2 <= alto y finalizar función hacer-mientras.

**Fase de Diseño:**

|  |
| --- |
| **ENTIDAD QUE RESUELVE EL PROBLEMA: lienzo** |
| **VARIABLES:**  X, Y: coordenadas  ancho, alto, distancia: entero |
| **NOMBRE ALGORITMO:** Dibujar\_escalera+punto  **INICIO ALGORITMO:**   1. anchoLienzo 🡨 600 2. altoLienzo 🡨 600 3. anchoC 🡨 40 4. distancia 🡨 30 5. x 🡨 0 6. y 🡨 100 7. y2 🡨 80 8. ***Hacer*** 9. *Dibujar* línea donde un extremo sea en (x, y) y el otro en (anchoLienzo, y) 10. y 🡨 y + (altoLienzo / 6) 11. ***Mientras*** y <= altoLienzo 12. ***Hacer*** 13. x2 🡨 distancia 14. ***Hacer*** 15. *Dibujar* circulo en (x2, y2) de (anchoC) 16. x2 🡨 x2 + (distancia \* 2) 17. ***Mientras*** x2 <= anchoLienzo 18. y2 🡨 y2 + (altoLienzo / 3) 19. ***Mientras*** y2 <= altoLienzo   **FIN ALGORITMO** |

# **Conclusión**

En esta segunda parte del trabajo practico 01 desarrollé la sección Sección Análisis – Diseño y Codificación de algoritmos – Aplicación de estructuras de control, donde puse en práctica la parte de análisis y diseño de algoritmo visto en la semana 1, el manejo de processing a la hora de realizar cálculos, mostrar ciertos objetos, aplicar ciertas funciones y a crear bucles con las estructuras iterativas MIENTRA, HACER-MIENTRAS y PARA.

Ésta segunda parte del TP me ayudó a practicar, entender y facilitar mi manejo con las estructuras Iterativas, el manejo de processing con algunas de sus distintas funciones y el cómo realizar el análisis y diseño de algoritmo de distintas situaciones.

# **Fuentes bibliográficas**

[07 Fases en la resolución de problemas mediante algoritmos](https://virtual.unju.edu.ar/mod/resource/view.php?id=299537) – Unju Virtual – Aula de Fundamentos de Programación Orientado a Objetos.

[14 Estructuras de Control Iterativas](https://virtual.unju.edu.ar/mod/resource/view.php?id=300666) – Unju Virtual – Aula de Fundamentos de Programación Orientado a Objetos.

[16 Programación con Processing](https://virtual.unju.edu.ar/mod/resource/view.php?id=306843) – Unju Virtual – Aula de Fundamentos de Programación Orientado a Objetos.

[Glosario funciones y términos comunes Processing](https://virtual.unju.edu.ar/mod/resource/view.php?id=306846) – Unju Virtual – Aula de Fundamentos de Programación Orientado a Objetos.

[Libro Hola Mundo con Processing](https://virtual.unju.edu.ar/mod/resource/view.php?id=306847) – Unju Virtual – Aula de Fundamentos de Programación Orientado a Objetos.

[PROCESSING: 4.6 Bucles while [En español]](https://youtu.be/dysxlptjAx0?si=y_e7swdSDyTz5My9) – YouTube – Canal: Air Room

[PROCESSING: 4.7 Bucles do while [En español]](https://youtu.be/4RQV0gMl8y8?si=enLQg-tlKig-7rBI) – YouTube – Canal: Air Room

[PROCESSING: 4.8 bucles for [En español]](https://youtu.be/kApd8u4l0tU?si=Chx2ABJGhZq-KRm6) – YouTube – Canal: Air Room

[Ejemplo análisis-diseño-codificación en Processing](https://youtu.be/_YblzDgoAus?si=pa2DMT7AlgVKCE-2) – YouTube – Canal: Ariel Vega

[Processing Tutorial -Mover una pelota- explicación fácil.](https://youtu.be/qQoPny-C1K0?si=HYyw_R8cUZpU6VUW) – YouTube – Canal: ANDRES MAURICIO PARRA LASPRILLA

[Processing - Movimiento de figuras con respecto al mouse](https://youtu.be/fYMV9PHcux0?si=1qUFW2xyDQRXLfx0) – YouTube – Canal: Juan Felipe Saldarriaga Colorado