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

Trabajo Práctico

N°1: Algoritmos y

Estructuras de Control

Tactaca, Cecilia Yasmin – LU TUV000764

*Profesores:*

*Mg. Ing. Ariel Alejandro Vega*

*Año 2024*

**FUNDAMENTOS DE PROGRAMACIÓN ORIENTADA A OBJETOS**

Índice

[Sección Expresiones aritméticas y lógicas 3](#_Toc163922762)

[Ejercicio 1: 3](#_Toc163922763)

[Desarrollo del ejercicio 1: 3](#_Toc163922764)

[Ejercicio 2: 3](#_Toc163922765)

[Desarrollo del ejercicio 2: 3](#_Toc163922766)

[Ejercicio 4: 3](#_Toc163922767)

[Ejercicio 5: 4](#_Toc163922768)

[Desarrollo del ejercicio 4-a: 4](#_Toc163922769)

[Desarrollo del ejercicio 4-b: 4](#_Toc163922770)

[Desarrollo del ejercicio 4-c: 4](#_Toc163922771)

[Ejercicio 6: 4](#_Toc163922772)

[Desarrollo del ejercicio 6: 4](#_Toc163922773)

[Ejercicio 7: 4](#_Toc163922774)

[Desarrollo del ejercicio 7: 4](#_Toc163922775)

[Ejercicio 8: 4](#_Toc163922776)

[Desarrollo del ejercicio 8: 4](#_Toc163922777)

[Ejercicio 9: 4](#_Toc163922778)

[Desarrollo del ejercicio 9: 4](#_Toc163922779)

[Ejercicio 10: 4](#_Toc163922780)

[Desarrollo del ejercicio 10: 5](#_Toc163922781)

[Ejercicio 11: 5](#_Toc163922782)

[Desarrollo del ejercicio 11: 5](#_Toc163922783)

[Sección Análisis – Diseño y Codificación de algoritmos – Aplicación de estructuras de control 5](#_Toc163922784)

[Ejercicio 12: 5](#_Toc163922785)

[Ejercicio 13: 5](#_Toc163922786)

[Ejercicio 14: 5](#_Toc163922787)

[Ejercicio 15: 5](#_Toc163922788)

[Ejercicio 16: 5](#_Toc163922789)

[Ejercicio 17: 5](#_Toc163922790)

[Ejercicio 18: 6](#_Toc163922791)

[Ejercicio 19: 6](#_Toc163922792)

[Ejercicio 20: 7](#_Toc163922793)

[Desarrollo del punto 20 7](#_Toc163922794)

[Ejercicio 21: 8](#_Toc163922795)

[Ejercicio 22: 9](#_Toc163922796)

[Conclusión 9](#_Toc163922797)

[Fuentes bibliográficas 9](#_Toc163922798)

# Sección Expresiones aritméticas y lógicas

Resolver cada ejercicio en un archivo Word y luego programarlo en Processing. En el caso de la programación crear un archivo por ejercicio.

Ejercicio 1: Evaluar (obtener resultado) la siguiente expresión para A = 2 y B = 5

### Desarrollo del ejercicio 1:

3\* A - 4 \* B / A ^ 2

(3\*A)-(4\*B/(A^2))

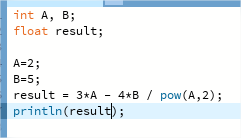
(3\*2)-(4\*5/(2^2))

6-(4\*5/4)

6-5

1

Captura de Processing



Ejercicio 2: Evaluar la siguiente expresión 4 / 2 \* 3 / 6 + 6 / 2 / 1 / 5 ^ 2 / 4 \* 2

### Desarrollo del ejercicio 2:

(((4/2)\*3)/6)+((((6/2)/1)/(5^ 2)) / 4) \* 2)

(((4/2)\*3)/6)+((((6/2)/1)/25) / 4) \* 2)

((2\*3)/6)+(((3/1)/25)/4)\*2)

(6/6)+((3/25)/4)\*2)

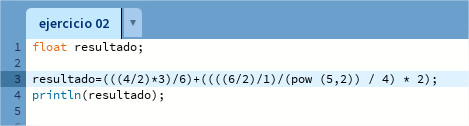
1+((0.12/4)\*2)

1+(0.03\*2)

1+0.06

1.06

Captura de Processing



Ejercicio 4: Evaluar las siguientes expresiones aritméticas, para lo cual indicar en el caso de las variables, el valor indicado. Luego escribirlas como expresiones algebraicas.

a) b ^ 2 – 4 \* a \* c

b) 3 \* X ^ 4 – 5 \* X ^ 3 + X \*12 – 17

c) (b + d) / (c + 4)

d) (x ^ 2 + y ^ 2) ^ (1 / 2)

### Desarrollo del ejercicio 4:

1. a=1, b=2 y c=3

b ^ 2 – ((4 \* a) \* c)

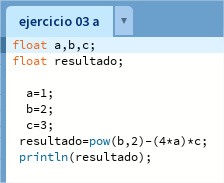
2 ^ 2 – ((4 \* 1 )\* 3)

4-((4\*1)\*3)

4-(4\*3)

4-12

-8



1. x=2

(3 \* (X ^ 4)) – (5 \*( X ^ 3 ))+ (X \* 12) – 17

(3 \* (2 ^ 4)) – (5 \*( 2 ^ 3 ))+ (2 \* 12) – 17

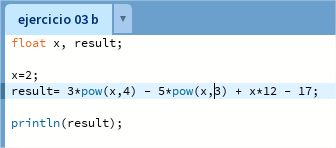
(3 \* 16) - (5\*8) + 24 -17

48 – 40 + 24 – 17

8 + 24 – 17

32 – 17

15



1. b=7, c=4 y d=9

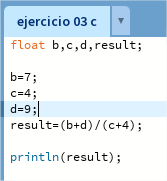
(b + d) / (c + 4)

(7+9)/(4+4)

16/(4+4)

16/8

2



1. x=3 e y=4

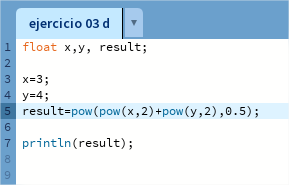
(x ^ 2 + y ^ 2) ^ (1 / 2)

(3 ^ 2 + 4 ^ 2) ^ (1 / 2)

(9 + 16) ^ (1 / 2)

(25) ^ (1 / 2)

5



Ejercicio 5: Si el valor de A es 4, el valor de B es 5 y el valor de C es 1, evaluar las siguientes expresiones:

1. B \* A – B ^ 2 / 4 \* C
2. (A \* B) / 3 ^ 2
3. (((B + C) / 2 \* A + 10) \* 3 \* B) – 6

### Desarrollo del ejercicio 5:

1. 5 \* 4 – 5 ^ 2 / 4 \* 1

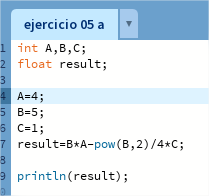
5\*4 – 25/4\*1

20 – 25/4\*1

20 – 25/4

20 – 6.25

13.75



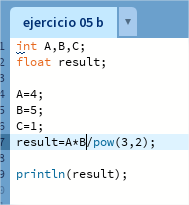
1. (A \* B) / 3 ^ 2

(4\*5) / 3 ^ 2

(4\*5) / 9

20/9

2.2222



1. (((B + C) / 2 \* A + 10) \* 3 \* B) – 6

(((5 + 1) / 2 \* 4 + 10) \* 3 \* 5) – 6

((6/2\*4+10)\*3\*5)-6

((3\*4+10)\*3\*5)-6

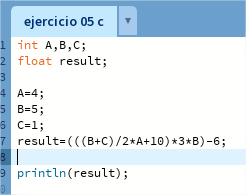
((12+10)\*3\*5)-6

(22\*3\*5)-6

(66\*5)-6

330-6

324



Ejercicio 6: Para x=3, y=4; z=1, evaluar el resultado de

R1 = y+z

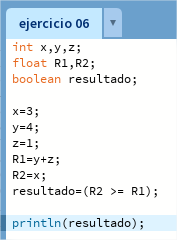
R2 = x >= R1

### Desarrollo del ejercicio 6:

R1=4+1=5

R2=3

R2>=R1 falso



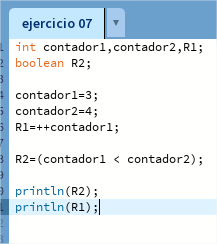
Ejercicio 7: Para contador1=3, contador2=4, evaluar el resultado de

R1 = ++contador1

R2 = contador1 < contador2

### Desarrollo del ejercicio 7:

R2= 4<4 = falso



Ejercicio 8: Para a=31, b=-1; x=3, y=2, evaluar el resultado de

a+b-1 < x\*y

### Desarrollo del ejercicio 8:

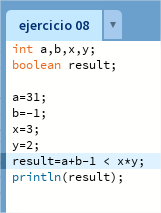
31+(-1)-1 < 3\*2

31+(-1)-1 < 6

30-1 < 6

29 < 6

Falso



Ejercicio 9: Para x=6, y=8, evaluar el resultado de

!(x<5)&& !(y>=7)

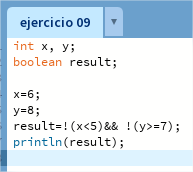
### Desarrollo del ejercicio 9:

!(6<5)&& !(8>=7)

¡falso && ¡verdadero

verdadero && falso

falso



Ejercicio 10: Para i=22, j=3, evaluar el resultado de

!((i>4) || !(j<=6))

### Desarrollo del ejercicio 10:

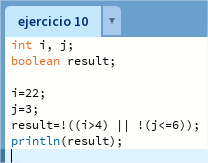
!((22>4) || !(3<=6))

!((verdadero) || !(verdadero))

!(verdadero || falso)

!(verdadero)

Falso



Ejercicio 11: Para a=34, b=12, c=8, evaluar el resultado de

!(a+b==c) || (c!=0)&&(b-c>=19)

### Desarrollo del ejercicio 11:

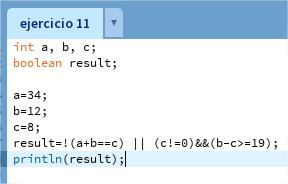
!(34+12==8) || (8!=0)&&(12-8>=19)

!(46==8) || (8!=0)&&(20>=19)

!(falso) || (verdadero)&&(verdadero)

verdadero || (verdadero)&&(verdadero)

verdadero



# 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 con un nombre ingresado por teclado.
* Análisis:

Datos de entrada:

Nombre.

Datos de salida:

Saludo con nombre.

Proceso:

¿Quién realiza el proceso? Processing

¿Cuál es el proceso que se realiza? Se pide que se ingrese un nombre, una vez ingresado el nombre por el teclado se emite en pantalla un saludo con el nombre.

#### Fase de diseño

|  |
| --- |
| Nombre de la entidad que resuelve el problema: Processing |
| Variables:  nombre: string // almacenan el nombre ingresado.  saludo: string // almacenan el saludo y nombre. |
| Nombre del algoritmo: hola\_nombre  Proceso del algoritmo   1. *inicio* 2. *Leer* nombre 3. *mensaje\_*saludo *← “Hola, ”*  nombre 4. *Mostrar* saludo 5. *fin* |

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: Calculo de perímetro y área de un rectángulo.
* Análisis:

Datos de entrada:

Base del rectángulo.

Altura de rectángulo.

Datos de salida:

Perímetro del rectángulo.

Área del rectángulo.

Proceso:

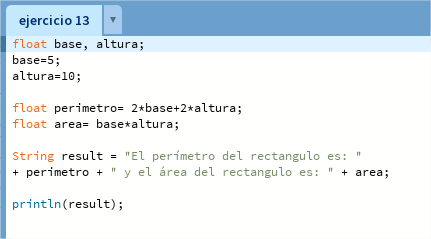
¿Quién realiza el proceso? Formula de perímetro y área de figuras geométricas.

¿Cuál es el proceso que se realiza? Se ingresan los datos de base y altura del rectángulo, luego se calcula el perímetro con la siguiente formula:

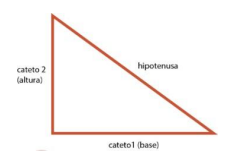
Para el calculo del área se usa la siguiente formula:

#### Fase de diseño

|  |
| --- |
| Nombre de la entidad que resuelve el problema: Formula matemática. |
| Variables:  base, altura: real // almacenan las medidas del rectángulo.  Perímetro, área: real // almacenan los resultados del perímetro y área del rectangulo. |
| Nombre del algoritmo: perímetro\_area  Proceso del algoritmo   1. *inicio* 2. *Leer* base 3. *Leer* altura 4. perimetro ← 2\*base + 2\*altura 5. area ← base \* altura 6. *Mostrar* “El perímetro del rectángulo es: “ + perímetro + “ y el área del rectángulo es: “ + area 7. *fin* |



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 hipotenusa de un triángulo rectángulo.
* Análisis:

Datos de entrada:

Cateto 1

Cateto 2

Datos de salida:

Hipotenusa

Proceso:

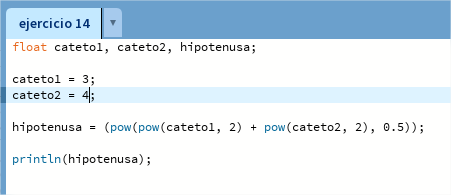
¿Quién realiza el proceso? Teorema de Pitágoras

¿Cuál es el proceso que se realiza? Al ser un triangulo rectángulo, uno de sus ángulos es de 90º, por ello se puede aplicar el teorema de Pitágoras para calcular sus catetos e hipotenusa.

Se ingresan las medidas de los catetos y mediante el teorema de Pitágoras se calcula la hipotenusa del triángulo rectángulo.

#### Fase de diseño

|  |
| --- |
| Nombre de la entidad que resuelve el problema: Teorema de Pitágoras |
| Variables:  cateto1, cateto2: real // almacenan las medidas de los catetos del triángulo.  hipotenusa: real // almacenan la hipotenusa calculada. |
| Nombre del algoritmo: calculo\_hipotenusa  Proceso del algoritmo   1. *Inicio* 2. *Leer* cateto1 3. *Leer* cateto2 4. hipotenusa *←* (a^2 + b^2 ) ^(0.5) 5. *Mostrar* hipotenusa 6. *Fin* |



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: Crear una calculadora que sume, reste, multiplique y divida dos números.
* Análisis:

Datos de entrada:

Número 1.

Número 2.

Datos de salida:

Resultado de la operación requerida.

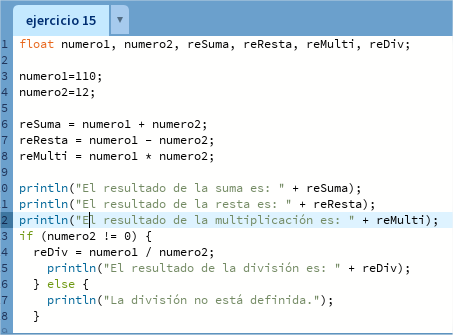
Proceso:

¿Quién realiza el proceso? Calculadora.

¿Cuál es el proceso que se realiza? Se ingresan dos números y sea suman, restan, multiplican y dividen entre ellos, y se muestra el resultado.

#### Fase de diseño

|  |
| --- |
| Nombre de la entidad que resuelve el problema: Calculadora. |
| Variables:  numero1, numero2: reales // almacenan los numero ingresados.  reSuma, reResta, reMulti, reDiv : real // almacenan el resultado de las operaciones |
| Nombre del algoritmo: calculadora  Proceso del algoritmo   1. *inicio* 2. *Leer* numero1 3. *Leer* numero2 4. reSuma ← numero1 + numero2 5. reResta ← numero1 – numero2 6. reMulti ← numero1 \* numero2 7. *mostrar ← “El resultado de la suma es: “* + reSuma 8. *mostrar ← “El resultado de la resta es: “* + reResta 9. *mostrar ← “El resultado de la multiplicación es: “* + reMulti 10. *Si* (numero2 !=0) *entonces* 11. reDiv ← numero1 / numero2 12. *mostrar ← “El resultado de la división es: “* + reDiv 13. *Si\_no* 14. *mostrar ← “La división no está definida.”* 15. *Fin* |



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: Convertir temperatura en grados Fahrenheit a grados Celsius.
* Análisis:

Datos de entrada:

Temperatura en grados Fahrenheit.

Datos de salida:

Temperatura en grados Celsius.

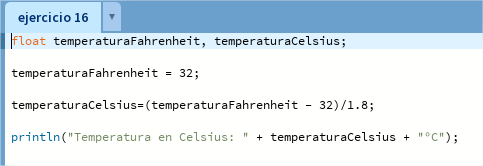
Proceso:

¿Quién realiza el proceso? Ecuación de conversión de unidades de temperatura.

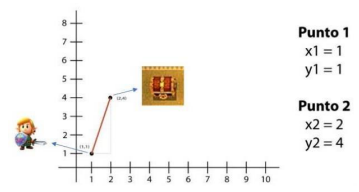
¿Cuál es el proceso que se realiza? Se ingresa la temperatura en grados Fahrenheit y mediante la ecuación de conversión de unidades se calcula la temperatura en grados Celsius.

#### Fase de diseño

|  |
| --- |
| Nombre de la entidad que resuelve el problema: Ecuación de conversión de unidades. |
| Variables:  temperaturaFahrenheit // almacenan la temperatura en grados Fahrenheit.  temperaturaCelsius // almacenan la temperatura en grados Celsius. |
| Nombre del algoritmo: temperatura  Proceso del algoritmo   1. *inicio* 2. *Leer* temperaturaFahrenheit 3. temperaturaCelsius ← (temperaturaFahrenheit – 32)/1.8 4. *Mostrar* “Temperatura en Celsius: “ + temperaturaCelsius + “ºC” 5. Fin |



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 lLnk con un Circulo, y al tesoro con un cuadrado. Además, mueva a Link mediante el mouse.

#### Fase de análisis

* Especificación del problema: Dibujar rectángulos rojos en un lienzo amarillo.
* Análisis:

Datos de entrada:

Tamaño del lienzo: alto y ancho en pixeles.

Tamaño de los rectángulos: ancho y alto en pixeles, de cada rectángulo.

Distancia entre los rectángulos.

Distancia entre los rectángulos y los límites del lienzo.

Datos de salida:

Lienzo amarillo con rectángulos rojos distribuidos como en la figura 1.

Proceso:

¿Quién realiza el proceso? Processing

¿Cuál es el proceso que se realiza? Se ingresan las medidas del lienzo y las medidas de los rectángulos que desean dibujarse, además se especifica su color. Posteriormente se calcula en que coordenadas deben dibujarse cada rectángulo con el uso de la estructura de control repetitiva for.

#### Fase de diseño

|  |
| --- |
| Nombre de la entidad que resuelve el problema: Processing |
| Variables:  alto, ancho: entero // almacenan las medidas del rectángulo.  anchoLienzo y altoLienzo // almacenan en el comando size.  distx, disty: entero // almacenan las distacias que hay entre cada rectángulo y los límites del lienzo.  coordenadas: vector // almacena las coordenadas donde se dibuja el rectángulo. |
| Nombre del algoritmo: ejercicio\_20  Proceso del algoritmo   1. inicio 2. anchoLienzo <-440 3. altoLienzo<-420 4. ancho<-40 5. alto<-20 6. distx<-20 7. disty<-20 8. coordenadas<-(distx, disty) 9. Para (x= cooredenadas.x) hasta (anchoLienzo) con paso (ancho+distx) Hacer 10. Para (y =coordenadas.y) hasta (altoLienzo) con paso (alto+disty) Hacer 11. Dibujar rectángulo en (x,y,ancho,alto) 12. Fin-para 13. Fin-para 14. Fin |

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: Calcular las raíces de una ecuación cuadrática.
* Análisis:

Datos de entrada:

Ecuación cuadrática.

Datos de salida:

Raíces de la ecuación.

Proceso:

¿Quién realiza el proceso? Processing

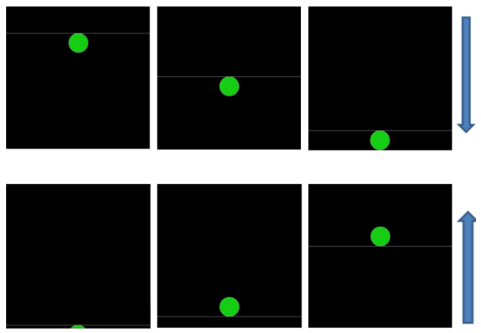
¿Cuál es el proceso que se realiza? Se ingresan las medidas del lienzo y las medidas de los rectángulos que desean dibujarse, además se especifica su color. Posteriormente se calcula en que coordenadas deben dibujarse cada rectángulo con el uso de la estructura de control repetitiva for.

#### Fase de diseño

|  |
| --- |
| Nombre de la entidad que resuelve el problema: Processing |
| Variables:  alto, ancho: entero // almacenan las medidas del rectángulo.  anchoLienzo y altoLienzo // almacenan en el comando size.  distx, disty: entero // almacenan las distacias que hay entre cada rectángulo y los límites del lienzo.  coordenadas: vector // almacena las coordenadas donde se dibuja el rectángulo. |
| Nombre del algoritmo: ejercicio\_20  Proceso del algoritmo   1. inicio 2. anchoLienzo <-440 3. altoLienzo<-420 4. ancho<-40 5. alto<-20 6. distx<-20 7. disty<-20 8. coordenadas<-(distx, disty) 9. Para (x= cooredenadas.x) hasta (anchoLienzo) con paso (ancho+distx) Hacer 10. Para (y =coordenadas.y) hasta (altoLienzo) con paso (alto+disty) Hacer 11. Dibujar rectángulo en (x,y,ancho,alto) 12. Fin-para 13. Fin-para 14. Fin |

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. El lienzo debería verse como en las siguientes figuras:



#### Fase de análisis

* Especificación del problema: Dibujar rectángulos rojos en un lienzo amarillo.
* Análisis:

Datos de entrada:

Tamaño del lienzo: alto y ancho en pixeles.

Tamaño de los rectángulos: ancho y alto en pixeles, de cada rectángulo.

Distancia entre los rectángulos.

Distancia entre los rectángulos y los límites del lienzo.

Datos de salida:

Lienzo amarillo con rectángulos rojos distribuidos como en la figura 1.

Proceso:

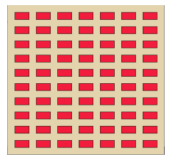
¿Quién realiza el proceso? Processing

¿Cuál es el proceso que se realiza? Se ingresan las medidas del lienzo y las medidas de los rectángulos que desean dibujarse, además se especifica su color. Posteriormente se calcula en que coordenadas deben dibujarse cada rectángulo con el uso de la estructura de control repetitiva for.

#### Fase de diseño

|  |
| --- |
| Nombre de la entidad que resuelve el problema: Processing |
| Variables:  alto, ancho: entero // almacenan las medidas del rectángulo.  anchoLienzo y altoLienzo // almacenan en el comando size.  distx, disty: entero // almacenan las distacias que hay entre cada rectángulo y los límites del lienzo.  coordenadas: vector // almacena las coordenadas donde se dibuja el rectángulo. |
| Nombre del algoritmo: ejercicio\_20  Proceso del algoritmo   1. inicio 2. anchoLienzo <-440 3. altoLienzo<-420 4. ancho<-40 5. alto<-20 6. distx<-20 7. disty<-20 8. coordenadas<-(distx, disty) 9. Para (x= cooredenadas.x) hasta (anchoLienzo) con paso (ancho+distx) Hacer 10. Para (y =coordenadas.y) hasta (altoLienzo) con paso (alto+disty) Hacer 11. Dibujar rectángulo en (x,y,ancho,alto) 12. Fin-para 13. Fin-para 14. Fin |

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:



### Desarrollo del punto 20

#### Fase de análisis

* Especificación del problema: Dibujar rectángulos rojos en un lienzo amarillo.
* Análisis:

Datos de entrada:

Tamaño del lienzo: alto y ancho en pixeles.

Tamaño de los rectángulos: ancho y alto en pixeles, de cada rectángulo.

Distancia entre los rectángulos.

Distancia entre los rectángulos y los límites del lienzo.

Datos de salida:

Lienzo amarillo con rectángulos rojos distribuidos como en la figura 1.

Proceso:

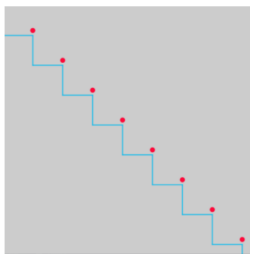
¿Quién realiza el proceso? Processing

¿Cuál es el proceso que se realiza? Se ingresan las medidas del lienzo y las medidas de los rectángulos que desean dibujarse, además se especifica su color. Posteriormente se calcula en que coordenadas deben dibujarse cada rectángulo con el uso de la estructura de control repetitiva for.

#### Fase de diseño

|  |
| --- |
| Nombre de la entidad que resuelve el problema: Processing |
| Variables:  alto, ancho: entero // almacenan las medidas del rectángulo.  anchoLienzo y altoLienzo // almacenan en el comando size.  distx, disty: entero // almacenan las distacias que hay entre cada rectángulo y los límites del lienzo.  coordenadas: vector // almacena las coordenadas donde se dibuja el rectángulo. |
| Nombre del algoritmo: ejercicio\_20  Proceso del algoritmo   1. inicio 2. anchoLienzo <-440 3. altoLienzo<-420 4. ancho<-40 5. alto<-20 6. distx<-20 7. disty<-20 8. coordenadas<-(distx, disty) 9. Para (x= cooredenadas.x) hasta (anchoLienzo) con paso (ancho+distx) Hacer 10. Para (y =coordenadas.y) hasta (altoLienzo) con paso (alto+disty) Hacer 11. Dibujar rectángulo en (x,y,ancho,alto) 12. Fin-para 13. Fin-para 14. Fin |

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: Dibujar escalones sobre un lienzo y colocar un punto rojo sobre cada escalón, como en la figura anterior.
* Análisis:

Datos de entrada:

PuntoA, PuntoB, PuntoC, puntos para definir las líneas.

PuntoD, punto para definir el punto rojo.

Distancia de la línea.

Datos de salida:

Imagen del lienzo con escalones y puntos rojos

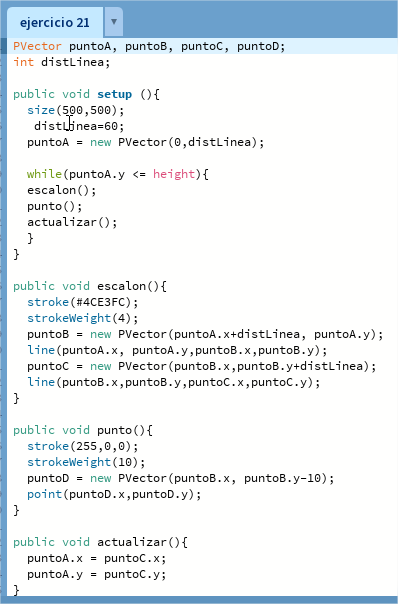
Proceso:

¿Quién realiza el proceso? Processing

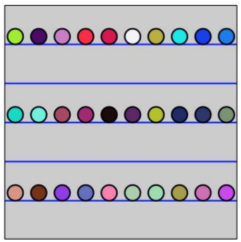
¿Cuál es el proceso que se realiza? Se dibuja una línea horizontal entre el puntoA y el puntoB, luego la línea vertical entre el puntoB y puntoC. Después se dibuja el punto rojo en la posición x del puntoB y en la posición y corrida 10 unidades de la posición y del puntoB. Se repite el proceso para llenar el lienzo.

#### Fase de diseño

|  |
| --- |
| Nombre de la entidad que resuelve el problema: Escalón. |
| Variables:  PuntoA, PuntoB, PuntoC, punto D // coordenadas cartesianas en 2D.  distLinea: entero// tamaño escalon. |
| Nombre del algoritmo: dibujar\_escalon  Proceso del algoritmo   1. *Inicio* 2. anchoLienzo ← 500 3. altoLienzo ← 500 4. distancia ← 60 5. *mientras* (puntoA.y sea menor o igual que anchoLienzo) *Hacer* 6. *dibujar* línea horizontal en (puntoA.x, puntoA.y, puntoB.x, puntoB.y) 7. *dibujar* línea vertical en (puntoB.x, puntoB.y, puntoC.x, puntoC.y) 8. dibujar punto en (puntoD.x, puntoD.y) 9. puntoA.x ← puntoC.x 10. puntoA.y ← puntoC.y 11. *fin\_mientras* 12. *Fin* |



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: Dibujar círculos en un lienzo de acuerdo a la distribución mostrada en la imagen.
* Análisis:

Datos de entrada:

Tamaño del lienzo: alto y ancho en pixeles.

Color de la línea.

Distancia entre las divisiones.

Datos de salida:

Lienzo dividido por líneas y círculos en algunas de ellas, como se muestra en la imagen.

Proceso:

¿Quién realiza el proceso? Processing

¿Cuál es el proceso que se realiza? Se dibuja el lienzo y se divide en franjas de igual medida, se dibujan líneas, y círculos alternadamente en las líneas. Los círculos están equidistantes y tienen coloreas aleatorios.

Fase de diseño

|  |
| --- |
| Nombre de la entidad que resuelve el problema: Processing |
| Variables:  distanciaCiruclo: int //almacena un valor entero  lineaX, lineaY, circuloX, circuloX, distanciaCirculo : int //almacena un valor entero  anchoLienzo, altoLienzo: int //almacenan valores enteros |
| Nombre del algoritmo: líneas\_circulos  Proceso del algoritmo   1. *inicio* 2. anchoLienzo ← 600 3. altoLienzo ← 600 4. lineaX ← 0 5. lineaY ← 100 6. distanciaCirculo ← 30; 7. circuloY ← 75 8. ***hacer*** 9. circuloX ← distanciaCirculo 10. ***hacer*** 11. *dibujar linea en* (lineaX, lineaY, anchoLienzo, lineaY) 12. *dibujar* circulo en circuloX, circuloY, 50, 50) 13. circuloX ← circuloX + distanciaCirculo\*2 14. ***fin\_hacer*** 15. ***mientras*** (circuloX sea menor que ancholienzo) 16. LineaY ← lineaY + 100; 17. circuloY ← circuloY + 200; 18. ***fin\_hacer*** 19. ***mientras***(lineaY sea menor que altoLienzo) 20. *fin* |

# Fuentes bibliográficas

<https://www.youtube.com/watch?v=EAclc8NcnOo&t=1s>

<https://www.youtube.com/watch?v=LUfoF-tG8xw>

<https://www.youtube.com/watch?v=Nr3NdAxjqsE>

<https://www.youtube.com/watch?v=_YblzDgoAus>

<https://www.youtube.com/watch?v=bMJhygO43cA>