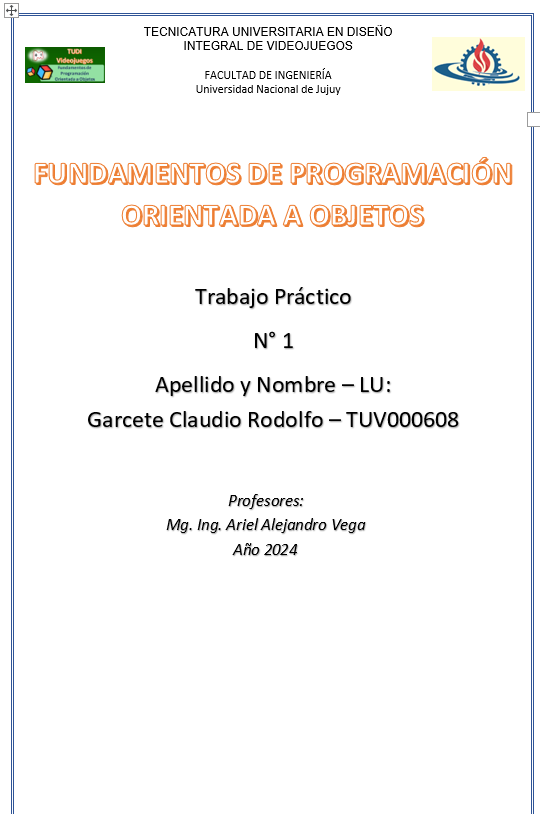
****

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

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

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

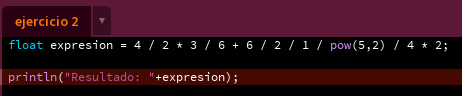
6/6+3/1/25/4\*2

1+3/25/4\*2

1+0.12/4\*2

1+0.06\*2

1+0.06 = 1.06





Ejercicio 3: Escribir las siguientes expresiones algebraicas como expresiones algorítmicas (en su forma aritmética dentro del algoritmo). En este caso no se pide evaluarlas ni programarlas.

No se hace.

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.**

1. b ^ 2 – 4 \* a \* c

b=4; c=1; a=2

b ^ 2 – 4 \* a \* c

4^ 2 – 4 \*2\*1

16-8\*1

16-8 = 8

1. 3 \* X ^ 4 – 5 \* X ^ 3 + X\*12 – 17

X=2

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 + d) / (c + 4)

b=20; d=30; c=21

(20 + 30) / (21+ 4)

50/25=2

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

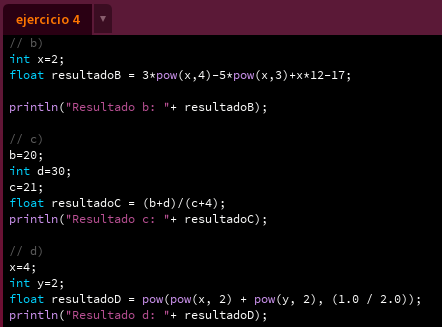
X=4; y=2

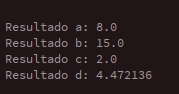
(4^2 + 2^ 2) ^ (1/2)

(16+4) ^ (1/2)

20^ (1/2)

20^ (0.5) = 4.47





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:

a) B \* A – B ^ 2 / 4 \* C

b) (A \* B) / 3 ^ 2

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

a) 5 \* 4 – 5 ^ 2 / 4 \* 1

20 - 25/4\*1

20 – 6.25\*1

20 - 6.25 = 13.75

b) (A \* B) / 3 ^ 2

(4\*5) / 3^ 2

20 / 9 = 2.22

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

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

((6 / 2 \* 4 + 10) \* 3 \* 5) – 6

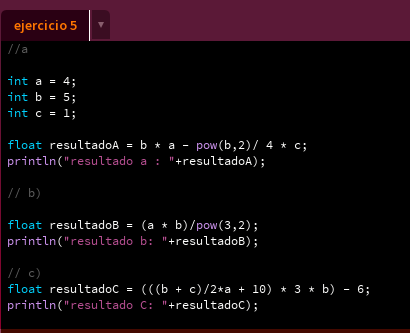
((3 \* 4 + 10) \* 3 \* 5) – 6

((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

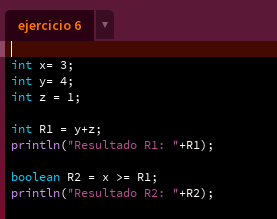
R2 = x >= R1

R1 = y+z

R1=4+1=5

R2 = x >= R1

R2 = 3 >= 5 = Falso





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

R1 = ++contador1

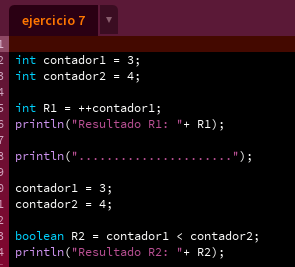
R2 = contador1 < contador2

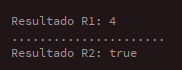
R1 = ++contador1

R1=4

R2 = contador1 < contador2

R2 =3<4 = Verdadero





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

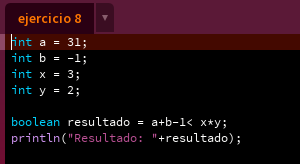
a+b-1< x\*y

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

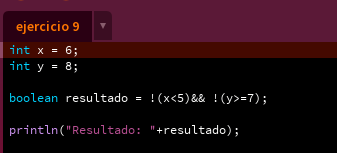
!(x<5)&& !(y>=7)

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

!Falso&& !Verdadero

Verdadero&&Falso

Falso





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

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

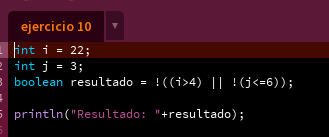
! ((22>4) || !(3<=6))

! ( (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)

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

!( 46==8) || (8!=0)&&(4>=19)

!(Falso) || (Verdadero)&&(Falso)

Verdadero || (Verdadero)&&(Falso)

Verdadero || Falso

Verdadero

# 

# 

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

Para cada ejercicio, en el archivo Word agregar las secciones de análisis y diseño, mientras que, para la codificación, crear el archivo de Processing.

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

Definición del problema: Ingresar por teclado el nombre del usuario y mostrar en pantalla con un saludo.

Análisis:

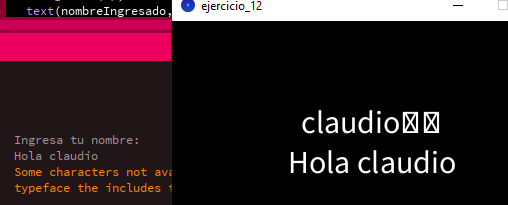
|  |  |
| --- | --- |
| Datos de entrada | nombreIngresado: Cadena |
| Proceso | Ingresar el nombre del usuario con foco en el lienzo y luego pulsar enter. |
| Datos de salida | Mostrar el nombre de un usuario y presentar en pantalla un saludo con el nombre indicado.  mensajeSaludo <- “Hola “+nombreIngresado |

Fase de diseño:

Fase de codificación

|  |
| --- |
| **Entidad que resuelve el problema:** Lienzo |
| **VARIABLES:**   * nombreIngresado : Cadena * mensajeSaludo: Cadena |
| **Nombre del algoritmo:** generarSaludo()  **Proceso del algoritmo:**  1- inicio  2- nombreIngresado <- “”  3- mensajeSaludo <- “”  4- Leer nombreIngresado  5- mensajeSaludo <- “Hola “ + nombreIngresado  6- Mostrar mensajeSaludo  7- 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

Definición del problema: Calcular el área y el perímetro de un rectángulo dada su base y su altura.

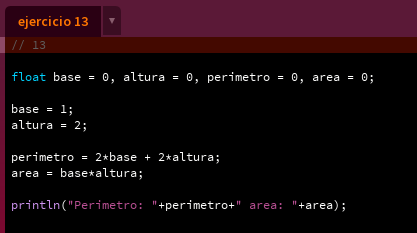
Análisis:

|  |  |
| --- | --- |
| Datos de entrada | base: Real  altura: Real |
| Proceso | perimetro <- 2\*base + 2\*altura  area <- base\*altura |
| Datos de salida | area: Real perimetro: Real |

Fase de diseño:

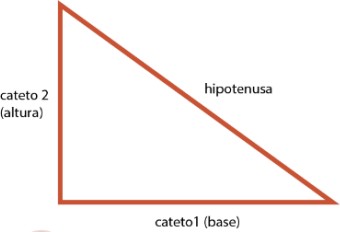
Fase de codificación

|  |
| --- |
| **Entidad que resuelve el problema:** Algoritmo |
| **VARIABLES:**   * base: Real * altura: Real * area: Real * perimetro: Real |
| **Proceso del algoritmo:**  1- inicio  2- Leer base  3- Leer altura  4- perímetro <- 2\*base + 2\*altura  5- area <- base\*altura  6- Mostrar “Perimetro: “+perímetro+” area: ”+área  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



Definición del problema: Obtener la hipotenusa de un triángulo rectángulo conociendo sus catetos

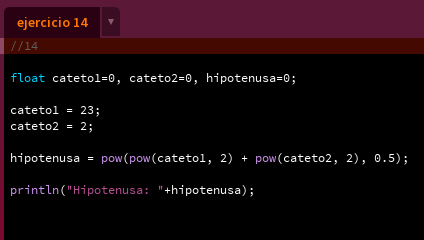
Análisis:

|  |  |
| --- | --- |
| Datos de entrada | cateto1: Real  cateto2: Real |
| Proceso | hipotenusa <- (cateto1 ^2 + cateto2 ^2) ^ (1/2) |
| Datos de salida | hipotenusa: Real |

Fase de diseño:

Fase de codificación

|  |
| --- |
| **Entidad que resuelve el problema:** Algoritmo |
| **VARIABLES:**   * cateto1: Real * cateto2: Real |
| **Proceso del algoritmo:**  1- inicio  2- Leer cateto1  3- Leer cateto2  4- hipotenusa <- (cateto1 ^2 + cateto2^2) ^ (1/2)  5- Mostrar “hipotenusa: “+hipotenusa  7- 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.

Definición del problema: Dado 2 números calcular: suma, resta, multiplicación y división.

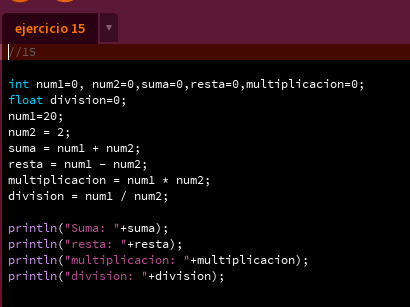
Análisis:

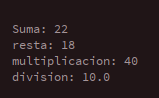
|  |  |
| --- | --- |
| Datos de entrada | num1: Entero  num2: Entero |
| Proceso | suma = num1 + num2;  resta = num1 - num2;  multiplicacion = num1 \* num2;  division = num1 / num2; |
| Datos de salida | suma,resta,multiplicación: Entero división: Real |

Fase de diseño:

Fase de codificación

|  |
| --- |
| **Entidad que resuelve el problema:** Calculadora |
| **VARIABLES:**   * num1, num2, suma, resta, multiplicacion: Entero * division: Real |
| **Proceso del algoritmo: suma**  1- inicio  2- Leer num1  3- Leer num2  4- suma <- num1 + num2;  5- Mostrar “Suma: “+Suma  7- fin |
| **Proceso del algoritmo: resta**  1- inicio  2- Leer num1  3- Leer num2  4- resta <- num1 - num2;  5- Mostrar “resta: “+resta  7- fin |
| **Proceso del algoritmo: multiplicacion**  1- inicio  2- Leer num1  3- Leer num2  4- multiplicacion <- num1 \* num2;  5- Mostrar “multiplicacion: “+multiplicacion  7- fin |
| **Proceso del algoritmo: division**  1- inicio  2- Leer num1  3- Leer num2  4- division <- num1 / num2;  5- Mostrar “division: “+division  7- 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



Definición del problema: convertir una temperatura Fahrenheit en grados Celsius

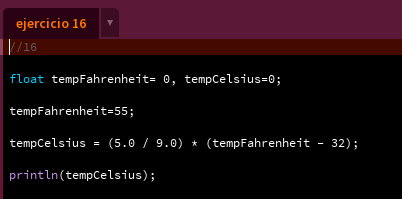
Análisis:

|  |  |
| --- | --- |
| Datos de entrada | tempFahrenheit: Real |
| Proceso | tempCelsius <- (5/9)\*( tempFahrenheit-32) |
| Datos de salida | tempCelsius: Real |

Fase de diseño:

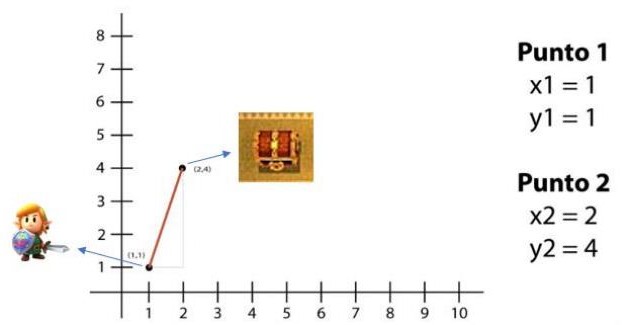
Fase de codificación

|  |
| --- |
| **Entidad que resuelve el problema:** Algoritmo |
| **VARIABLES:**   * tempFahrenheit, tempCelsius: Real |
| **Proceso del algoritmo:**  1- inicio  2- Leer tempFahrenheit  3- tempCelsius <- (5/9) \* (tempFahrenheit - 32)  4- mostrar tempCelsius  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 Link con un Circulo, y al tesoro con un cuadrado. Además, mueva a Link mediante el mouse.

Definición del problema: Hallar la distancia entre los objetos Link y tesoro.

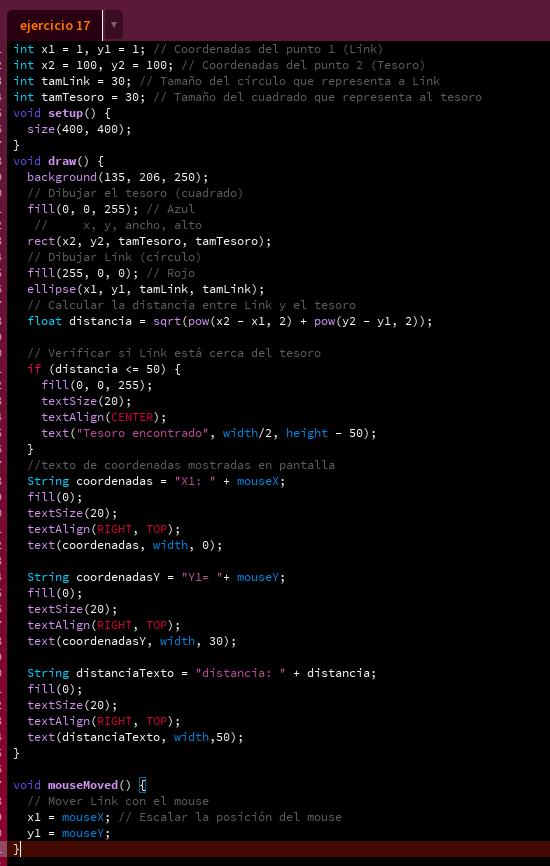
Análisis:

|  |  |
| --- | --- |
| Datos de entrada | Coordenadas de link y el tesoro |
| Proceso | Formula de distancia: d = ((x2-x1)^2 + (y2-y1)^2) ^(1/2) |
| Datos de salida | Distancia entre link y el tesoro |

Fase de diseño:

Fase de codificación

|  |
| --- |
| **Entidad que resuelve el problema:** Persona |
| **VARIABLES:**   * **x1: float //** almacena un valor decimal * **y1: float // almacena un valor decimal** * **x2: float //** almacena un valor decimal * **y2: float //** almacena un valor decimal * **coordenadaX:** float // almacena el resultado de un calculo * **coordenadaY:** float //almacena el resultado de un calculo * **distancia: float // almacena el resultado de un calculo** * **distanciaTesoro: float // almacena un valor** |
| **Proceso del algoritmo:**  ***Inicio***   1. *Leer x1* 2. *Leer y1* 3. *Leer x2* 4. *Leer y2* 5. *distanciaTesoro ← 50* 6. *coordenadaX ← x2 - x1* 7. *coordenadaY ← y2 – y1* 8. *distancia ← ((coordenadaX)^2 + (coordenadaY)^2)^2* 9. *mostrar “la distancia es de: ” + distancia* 10. ***si*** *(distancia = distanciaTesoro)* ***entonces*** 11. ***mostrar*** *“¡PowerUp activado!”* 12. *fin\_si*   ***fin*** |
| **Entidad que resuelve el problema:** Distancia |
| **VARIABLES:**   * **x1: float //** almacena un valor decimal * **y1: float // almacena un valor decimal** * **x2: float //** almacena un valor decimal * **y2: float //** almacena un valor decimal * **coordenadaX:** float // almacena el resultado de un calculo * **coordenadaY:** float //almacena el resultado de un calculo * **distancia: float // almacena el resultado de un calculo** * **distanciaTesoro: float // almacena un valor** |
| **Proceso del algoritmo:**  ***Inicio***   1. *Leer x1* 2. *Leer y1* 3. *Leer x2* 4. *Leer y2* 5. *distanciaTesoro ← 50* 6. *coordenadaX ← x2 - x1* 7. *coordenadaY ← y2 – y1* 8. *distancia ← ((coordenadaX)^2 + (coordenadaY)^2)^2* 9. *mostrar “la distancia es de: ” + distancia* 10. ***si*** *(distancia = distanciaTesoro)* ***entonces*** 11. ***mostrar*** *“¡PowerUp activado!”* 12. *fin\_si*   ***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.

Definición del problema: Obtener las raíces de una ecuación de segundo grado.

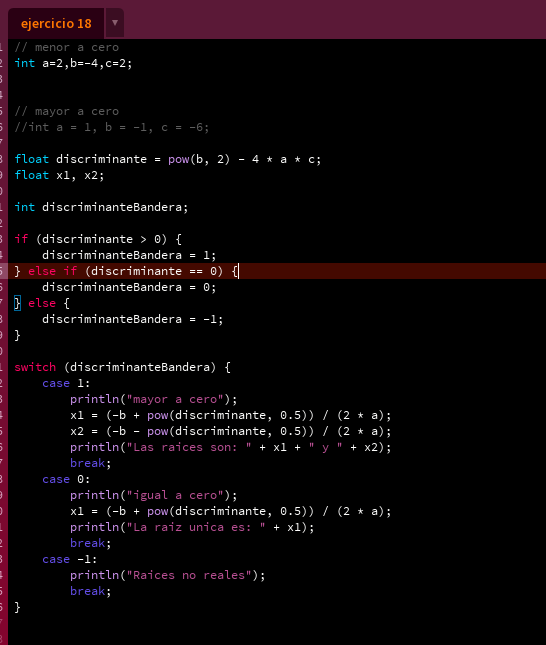
Análisis:

|  |  |
| --- | --- |
| Datos de entrada | a,b,c : Real |
| Proceso | Cálculo de raíces:  X1 <- (-b+ (b ^2-4\*a\*c) ^(1/2))/(2\*a)  X2<- (-b- (b ^2-4\*a\*c) ^(1/2))/(2\*a) |
| Datos de salida | Raíces de la ecuación cuadrática  x1, x2: Real |

Fase de diseño:

Fase de codificación

|  |
| --- |
| **Entidad que resuelve el problema:** Algoritmo |
| **VARIABLES:**  a, b, c, x1, x2, discriminante: Real |
| **Proceso del algoritmo:**  1- Inicio  2- Leer a  3- Leer b  4- Leer c  5- discriminante <- b^2-4\*a\*c  6- según\_sea (discriminante) hacer  7- caso discriminante > 0:  8- x1 <- (-b+ (b ^2-4\*a\*c) ^ (1/2)) /(2\*a)  9- x2 <- (-b- (b ^2-4\*a\*c) ^ (1/2)) /(2\*a)  10- Mostrar “Las raices son: “, x1, x2  11- caso discrimante=0:  12- x1 <- (-b+ (b ^2-4\*a\*c) ^ (1/2)) /(2\*a)  13- x2 <- (-b- (b ^2-4\*a\*c) ^ (1/2)) /(2\*a)  14- Mostrar “La única raiz es: “, x1  15- caso discrimante<0:  16- Escribir “Las raices no tienen solución real”  17- 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



Definición del problema: Obtener las raíces de una ecuación de segundo grado.

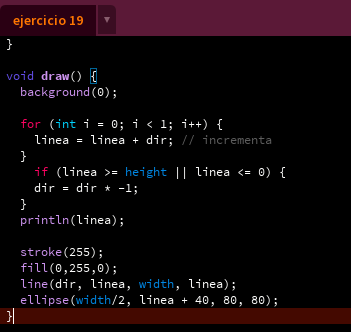
Análisis:

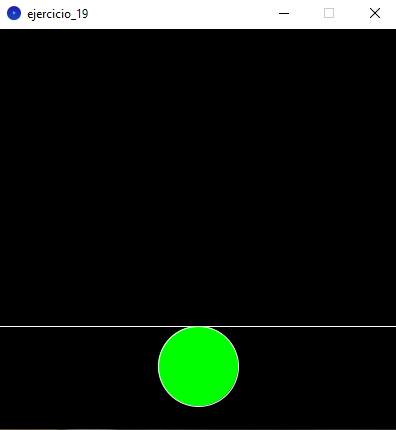
|  |  |
| --- | --- |
| Datos de entrada | Linea, dir : Real |
| Proceso | La computadora |
| Datos de salida | Bucle de la linea y circulo |

Fase de diseño:

Fase de codificación

|  |
| --- |
| **Entidad que resuelve el problema:** Lienzo |
| **VARIABLES:**   * **linea: entero //** almacena un valor entero   **dir : entero // almacena un valor enter** |
| **Proceso del algoritmo:**  *inicio*   1. *Leer linea* 2. *Leer dir* 3. *anchoLienzo ← 400* 4. *altoLienzo ← 400* 5. ***para*** *i ← 0 hasta* ***alto*** *incremento 1* ***hacer*** 6. *linea ← linea + dir* 7. ***fin\_para*** 8. *si ((linea >= anchoLienzo) O (linea <= 0))* ***entonces*** 9. *dir ← dir \* (-1)* 10. ***fin\_si*** 11. *mostrar linea* 12. *dibujar linea en (dir, linea, altoLienzo, linea)* 13. *dibujar circulo en (altoLienzo/2, linea + 40, 80, 80)*   *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:



Fase de análisis

Definición del problema: Dibujar rectángulos de 40 ancho y 20 de alto en un lienzo de (440, 220) y mantener una distancia de 20 pixeles tanto horizontal como vertical con estructuras iterativas.

Análisis:

**Datos de entrada**

altoRec: Entero

anchoRec: Entero

distRec: Entero

coordenadas: coordenadas cartesianas

**Datos de salida**

Se muestra en el lienzo rectángulos de 40x20 con una distancia horizontal y vertical de 20 pixeles. (Los rectángulos dibujados)

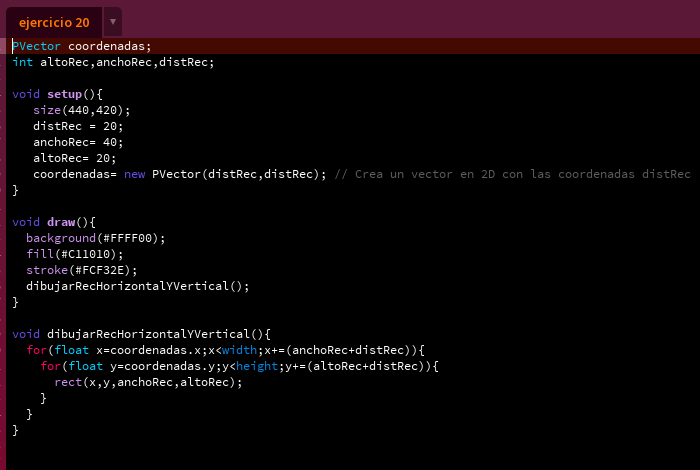
**Proceso**

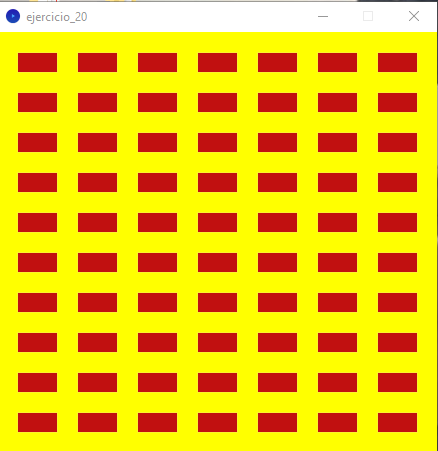
Realizar iteración anidado con estructura for. La anidación de una estructura for en otra, crea la iteración en dos dimensiones. (Dibujar los rectángulos)

Fase de diseño:

Fase de codificación

|  |
| --- |
| **Entidad que resuelve el problema:** Lienzo |
| **VARIABLES:**   * anchoRect: Entero * altoRect: Entero * distRect: Entero * coordenadas: Coordenadas * ancho, alto * anchoLienzo, altoLienzo: Entero |
| **Nombre del algoritmo:** dibujarRecHorizontalYVertical()  **Proceso del algoritmo:**  1- inicio  2- anchoLienzo <- 440  3- altoLienzo <- 420  4- distRec <- 20  5- anchoRec <- 40  6- altoRec <- 20  7- coordenadas <- new PVector(distRec, distRec)  8- Para x <- coordenadas.x hasta width con paso (anchoRec+distRec)  9- Hacer  10- para y <- coordenadas.x hasta height con paso (altoRec+distRec)  11-Hacer  12-rect(x,y,anchoRec,altoRec)  13-fin\_para  14-fin\_para  15- fin |





**Ejercicio 21:** Utilizando la estructura de control repetitiva while() dibuje la siguiente imágenes 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

Definición del problema: Dibujar escalones sobre el lienzo y colocar sobre cada escalon un punto rojo.

Análisis:

**Datos de entrada**

puntoA, puntoB, puntoC, puntoD: Entero

**Datos de salida**

El dibujo de la línea horizontal

El dibujo de la línea vertical

El dibujo del punto rojo

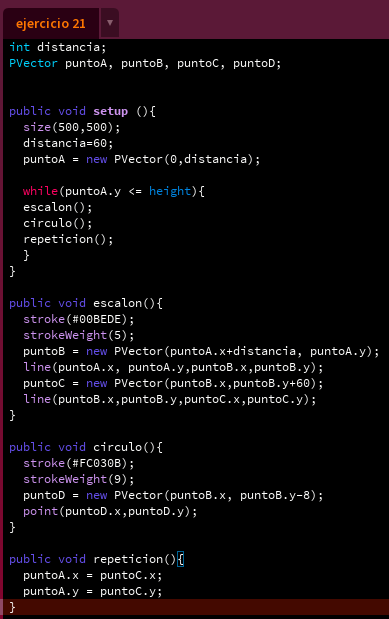
**Proceso**

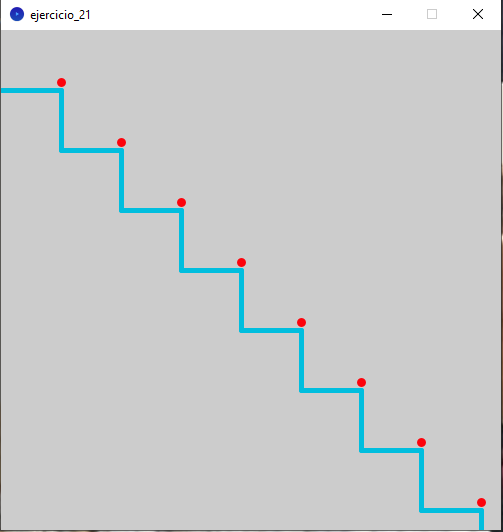
* Dibujar una línea horizontal entre los puntos A y B
* Dibujar una línea vertical entre los puntos B y C
* Dibujar un punto en la siguiente posición: x = posición en x de B, y = posición en y de B – 5 unidades.
* Actualizar las coordenadas de puntoA con las de puntoC
* Repetir desde el principio hasta que la coordenada en de puntoA sea mayor que el alto del lienzo

Fase de diseño:

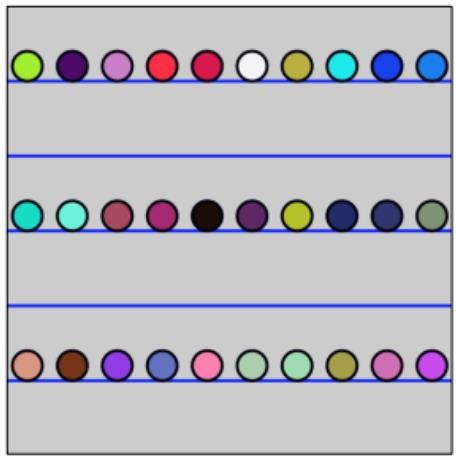
Fase de codificación

|  |
| --- |
| **Entidad que resuelve el problema:** Escalón |
| **VARIABLES:**  puntoA,puntoB,puntoC,puntoD: int //almacena un vector  distancia: int //almacena un valor entero |
| **Nombre del algoritmo:** dibujarEscalon()   1. **Proceso del algoritmo:** *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 circulo en (puntoD.x, puntoD.y)* 9. *puntoA.x ← puntoC.x* 10. *puntoA.y ← puntoC.y* 11. ***fin\_mientras***   *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

Definición del problema: Dibujar 5 lineas en el lienzo y 10 circulos sobre cada línea de por medio.

Análisis:

**Datos de entrada**

lineaX, lineaY, circuloX, circuloY,distanciaCirculo: Entero

**Datos de salida**

5 lineas horizontales.

10 circulos sobre cada línea de por medio.

**Proceso**

* Dibujar 5 líneas.
* Dibujar 10 circulos cada linea de por medio.

Fase de diseño:

Fase de codificación

|  |
| --- |
| **Entidad que resuelve el problema:** Lienzo |
| **VARIABLES:**  lineaX, lineaY, circuloX, circuloY,distanciaCirculo: Entero |
| **Nombre del algoritmo:** dibujar\_lineas\_y\_circulos ()  **Proceso del algoritmo:** 1- Inicio  2- lineaX<-0  3- lineaY<-100  4- circuloX<-0  5- circuloY<-75  6- distanciaCirculo <- 30  7- **hacer**  8- circuloX<- distanciaCirculo  9- dibujar linea(lineaX,lineaY,width,lineaY);  10- **hacer**  11- dibujar circulo (circuloX,circuloY,50,50)  12- circuloX += distanciaCirculo\*2  13- **fin\_hacer**  14- **mientras** (circuloX < width)  15- lineaY += 100;  16- circuloY += 200;  17- **fin\_hacer**  18- **mientras** (lineaY<=height);  19-Fin |

