

FUNDAMENTOS DE PROGRAMACIÓN

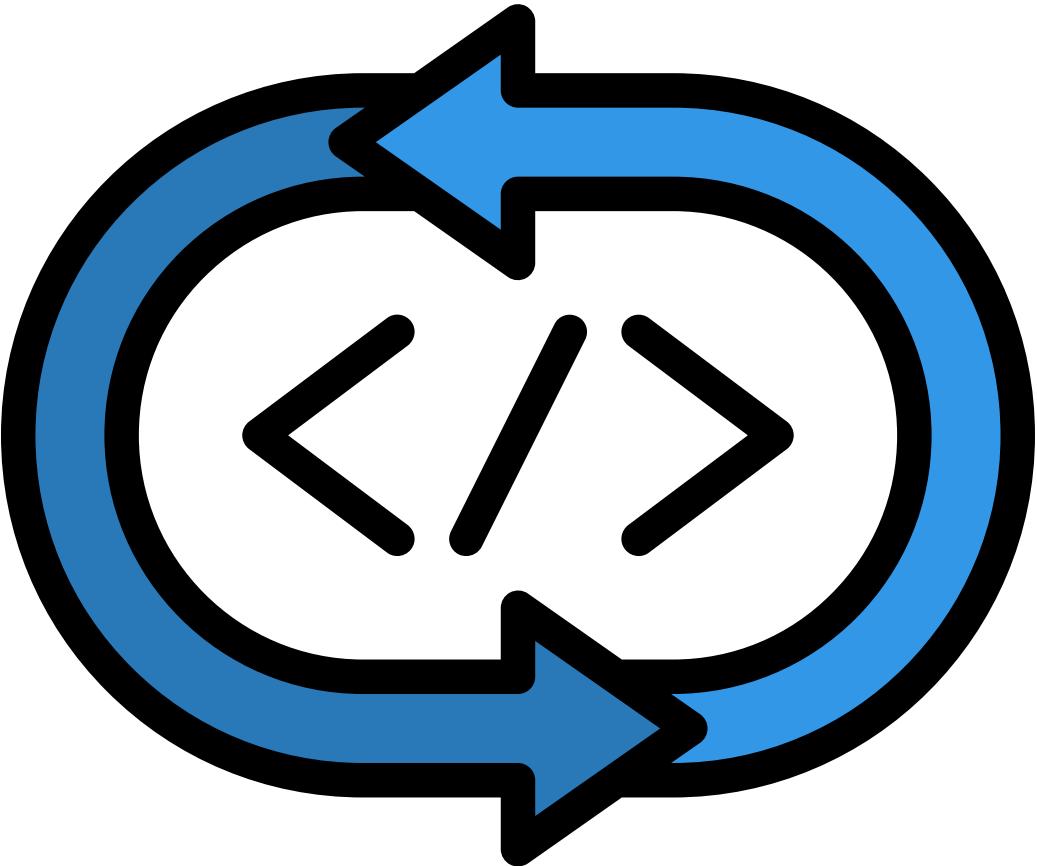


FUNDAMENTOS DE PROGRAMACIÓN



¿QUÉ VAMOS A VER?

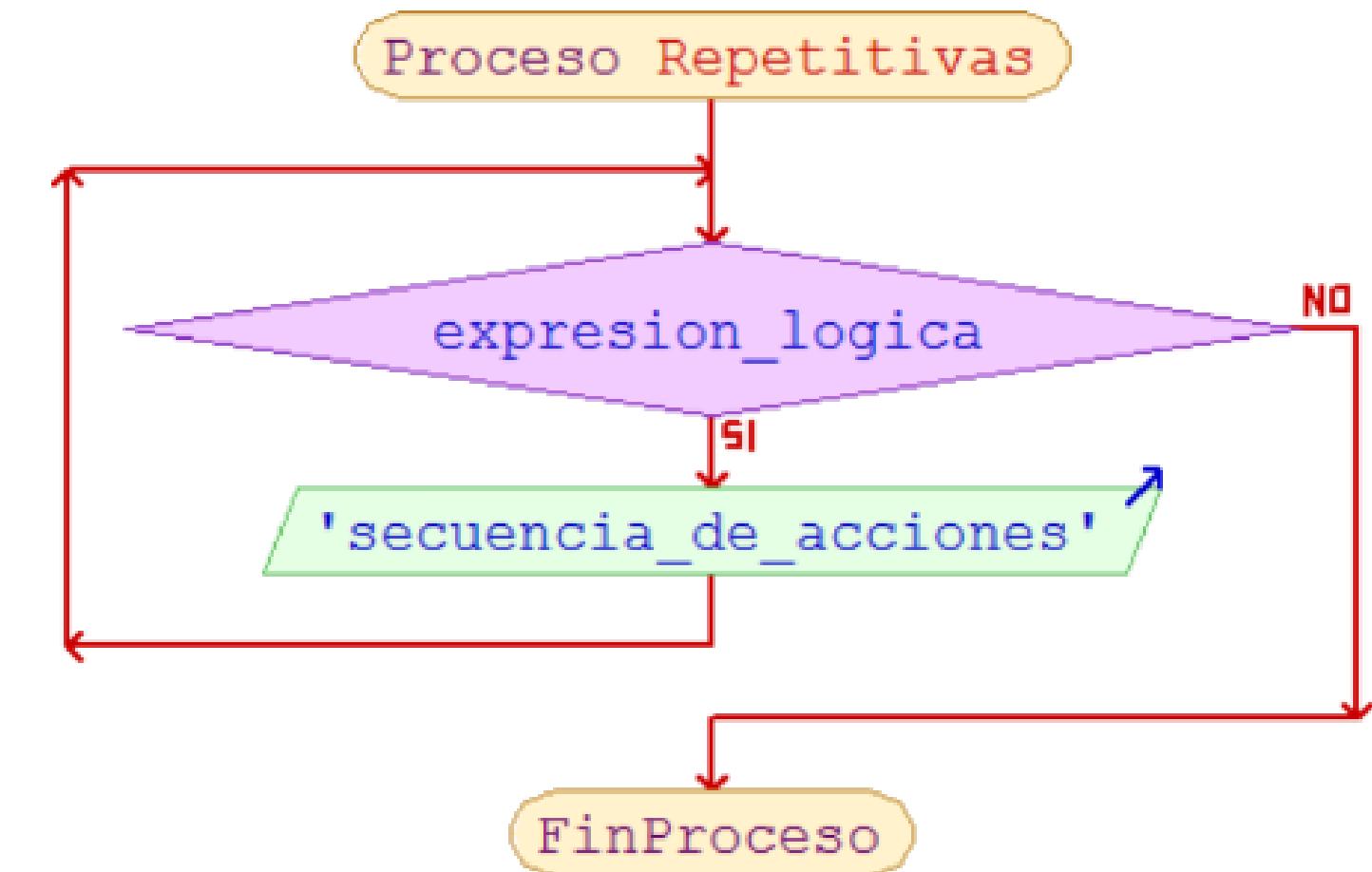
- Estructuras de control repetitivas
- Contadores
- Acumuladores
- Estructura “Mientras”
- Estructura “Repetir”
- Para
- Patrones comunes





ESTRUCTURAS DE CONTROL REPETITIVAS

- Están diseñadas para que una expresión sea evaluada muchas veces.
- Son estructuras conocidas como bucles o ciclos.
- Nos permiten ejecutar una instrucción x cantidad de veces, dependiendo siempre del tipo de solución que implementemos.
- Existen tres estructuras repetitivas
 - **Mientras**
 - **Repetir... hasta**
 - **Para**
- El objetivo de las estructuras siempre es el mismo, lo que cambia es su sintaxis y forma de abordar la problemática.





CONTADORES

- Variable utilizada en las estructuras de control repetitivas.
- Es una variable de tipo entero.
- Incrementa o decrementa su valor de manera constante.
- Normalmente se utiliza para contar la cantidad de veces que se ejecuta una acción o se cumple una condición.

```
1  Algoritmo contador
2
3      Definir count Como Entero
4      count ← 0
5
6      Mientras count ≤ 5 Hacer
7          Escribir count
8          count ← count + 1
9      Fin Mientras
10
11 FinAlgoritmo|
```



ACUMULADORES

- Variable utilizada en las estructuras de control repetitivas.
- Es una variable numérica.
- Incrementa o decrementa su valor de manera no constante.
- Se utiliza para acumular valores en una sola variable, pudiendo ser por ejemplo pagos, cantidades variables, cantidad de productos, etc.

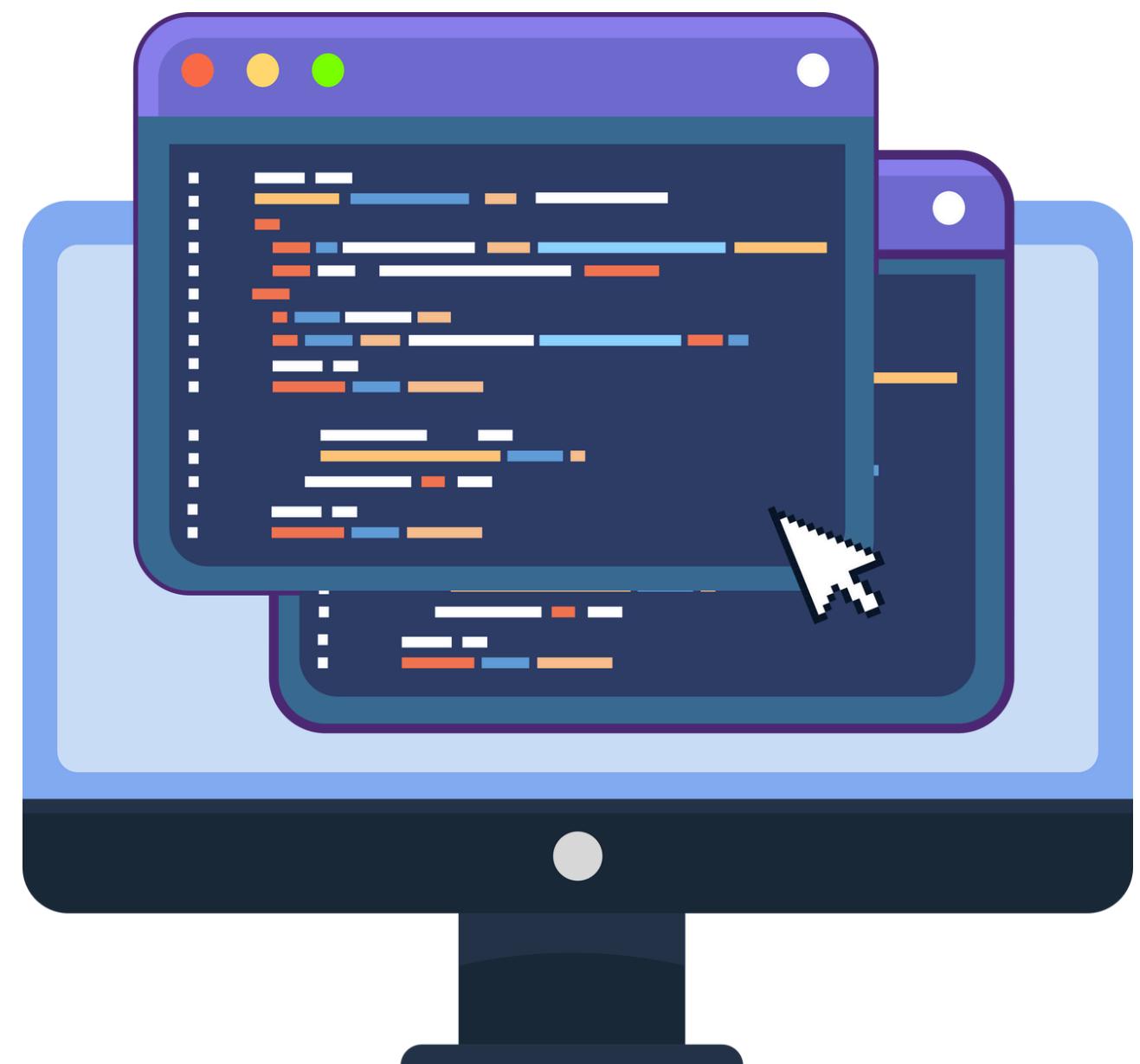




ESTRUCTURA “MIENTRAS”

- Repite el conjunto de instrucciones mientras la condición evaluada sea verdadera.
- Si la condición es falsa, el ciclo termina.
- En esta estructura, primero se evalúa la condición y según el resultado se entra o no en el bucle.

```
1 Algoritmo bucle  
2  
3   Mientras expresion_logica Hacer  
4     |   secuencia_de_acciones  
5   Fin Mientras  
6  
7 FinAlgoritmo
```





ESTRUCTURA “MIENTRAS”

EJEMPLO

- Realice un algoritmo que permita mostrar por pantalla la suma de los 20 primeros números naturales.

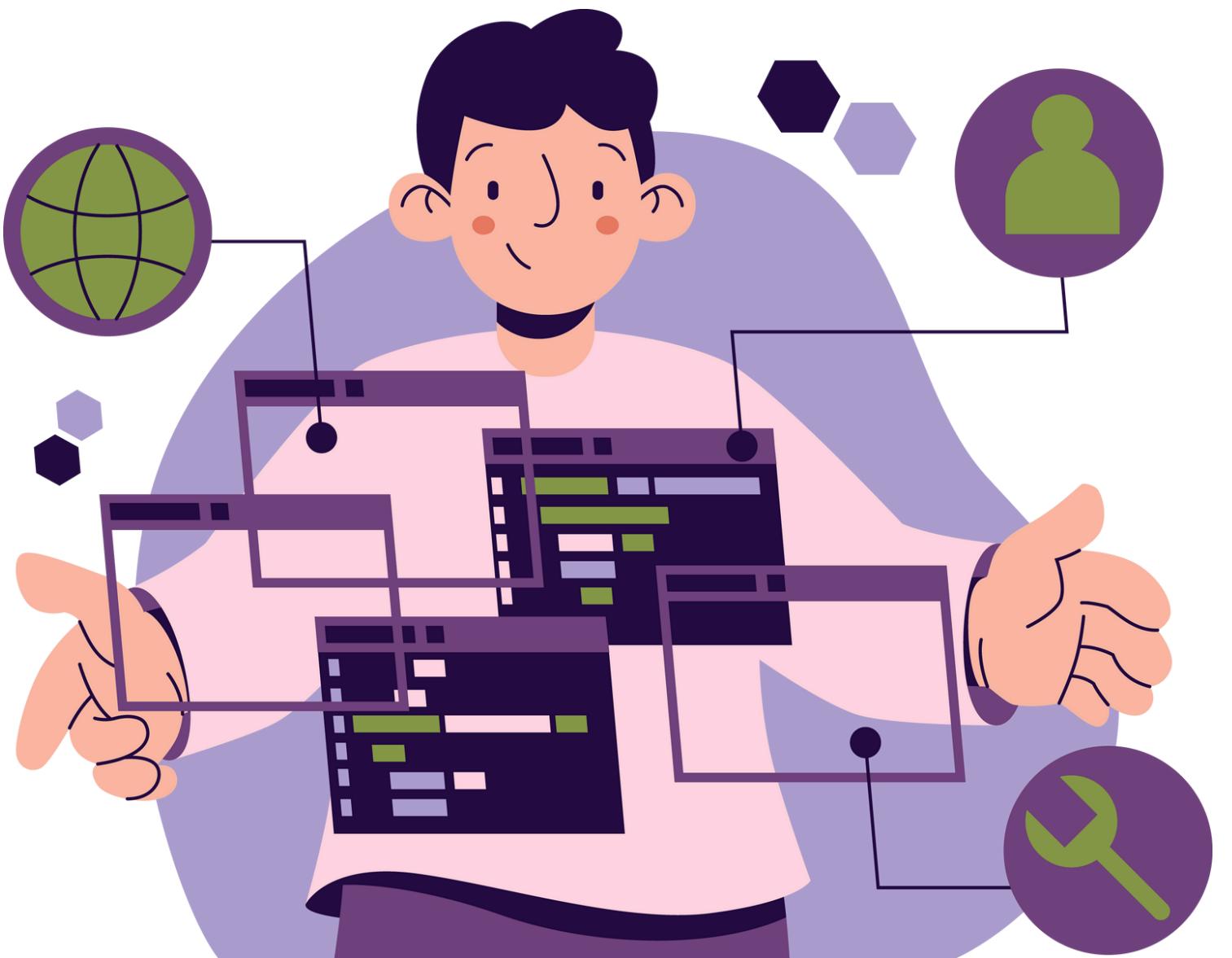
```
1 Algoritmo bucle
2
3 Definir acumulador Como Entero
4 Definir contador Como Entero
5
6 acumulador ← 0
7 contador ← 1
8
9 Mientras contador ≤ 20 Hacer
10     acumulador ← acumulador + contador
11     contador ← contador + 1
12 FinMientras
13
14 Escribir "La sumatoria es: " acumulador
15
16 FinAlgoritmo|
```



ESTRUCTURA “REPETIR”

- Esta estructura se repite hasta que la condición lógica resulta ser verdadera.
- Si la condición de termino es falsa, el ciclo vuelve al inicio.
- En este tipo de estructura el bloque de código se ejecuta como mínimo una vez.

```
1 Algoritmo bucle  
2  
3 Repetir  
4 | secuencia_de_acciones  
5 | Hasta Que expresion_logica  
6  
7 FinAlgoritmo
```





ESTRUCTURA “REPETIR”

EJEMPLO

- Realice un algoritmo que permita mostrar por pantalla la suma de los 20 primeros números naturales. Esta vez utilice una estructura “Repetir... Hasta”.

```
1 Algoritmo bucle
2
3 Definir acumulador Como Entero
4 Definir contador Como Entero
5
6 acumulador ← 0
7 contador ← 1
8
9 Repetir
10    acumulador ← acumulador + contador
11    contador ← contador + 1
12 Hasta Que contador > 20
13
14 Escribir "El resultado es: " acumulador
15
16 FinAlgoritmo
```



PARA

- Este tipo de ciclo repite el bloque de código una cantidad determinada de veces.
- Posee un valor de finalización de la variable de control del ciclo y otro valor de paso o incremento.
- La variable de control aumenta a medida que se repite el ciclo.
- El ciclo finaliza cuando la variable de control es la misma que el valor final. **Algoritmo bucle**

```
Para variable_numerica<-valor_inicial Hasta valor_final Con Paso paso Hacer
    secuencia_de_acciones
Fin Para

FinAlgoritmo
```



PARA

EJEMPLO

- Realice un algoritmo que permita mostrar por pantalla la suma de los 20 primeros números naturales. Esta vez utilice una estructura “Para”.

```
1 Algoritmo bucle
2
3 Definir acumulador Como Entero
4
5 acumulador ← 0
6
7 Para i<0 Hasta 20 Con Paso 1 Hacer
8     acumulador ← acumulador + i
9 Fin Para
0
1 Escribir "El resultado final es: " acumulador
2
3 FinAlgoritmo
```



PATRONES COMUNES

- Un patrón es una solución aplicable a una problemática que ocurre a menudo.
- Existen formas de abordar un problema que se pueden replicar en diversos contextos.





PATRONES COMUNES

SUMAR COSAS

- Escriba un programa que reciba como entrada un número entero. El programa debe mostrar el resultado de la suma de los números al cuadrado desde el 1 hasta el valor ingresado.

```
1 Algoritmo sumar_cosas
2
3   Escribir "Ingrese n: "
4   Leer n
5   suma ← 0
6   cont ← 1
7
8   Mientras cont ≤ n Hacer
9     d ← cont↑2
10    suma ← suma + d
11    cont ← cont + 1
12  FinMientras
13
14  Escribir "La suma de los números al cuadrado es: ", suma
15
16 FinAlgoritmo
```



PATRONES COMUNES

MULTIPLICAR COSAS

- Escriba un programa que calcule la factorial de un número n ingresada como entrada:

```
Algoritmo multiplicar_cosas
    Escribir "Ingrese n: "
    Leer n

    Si n < 0 Entonces
        Escribir "La factorial está definida sólo para números naturales mayores o igual que 0"
    SiNo
        prod = 1
        cont = 1
    FinSi

    Mientras cont ≤ n Hacer
        prod ← prod * cont
        cont ← cont + 1
    FinMientras

    Escribir "La factorial de ", n, " es: ", prod

FinAlgoritmo
```



PATRONES COMUNES

CONTAR COSAS

- Escriba un programa que solicite el ingreso de n números, luego entregue la cantidad de números pares ingresados.

```
1 Algoritmo contar_cosas
2
3   Escribir "Ingrese n: "
4   Leer n
5
6   pares ← 0
7   cont ← 0
8
9   Mientras cont < n Hacer
10      Escribir "Ingrese número: "
11      Leer num
12
13      Si num % 2 = 0 Entonces
14          pares ← pares + 1
15      FinSi
16
17      Escribir "pares = ", pares
18      cont ← cont + 1
19  FinMientras
20
21  Escribir "La cantidad de pares ingresados es: ", pares
22
23 FinAlgoritmo
```



PATRONES COMUNES

ENCONTRAR EL MÍNIMO

- ¿Cómo cambia el patrón?

```
1  Algoritmo encontrar_minimo
2
3      Escribir "Ingrese n: "
4      Leer n
5
6      i  $\leftarrow$  1
7      numMenor = 999999
8
9      Mientras i  $\leq$  n Hacer
10         Escribir "Ingrese un número: "
11         Leer num
12
13         Si numMenor > num Entonces
14             numMenor  $\leftarrow$  num
15         FinSi
16
17         Escribir "Menor temporalmente = ", numMenor
18         i  $\leftarrow$  i + 1
19     FinMientras
20
21     Escribir "El número menor es: ", numMenor
22
23 FinAlgoritmo
```



EJERCICIOS

INGRESO DE DATOS

- Tienes una clase de 10 estudiantes.
- Crea una estructura que te permita ingresar la edad de los 10 estudiantes.
- Cada vez que ingreses un dato debes comprobar si es mayor de edad.
- Al finalizar muestra por pantalla cuantos alumnos efectivamente tienen la mayoría de edad.

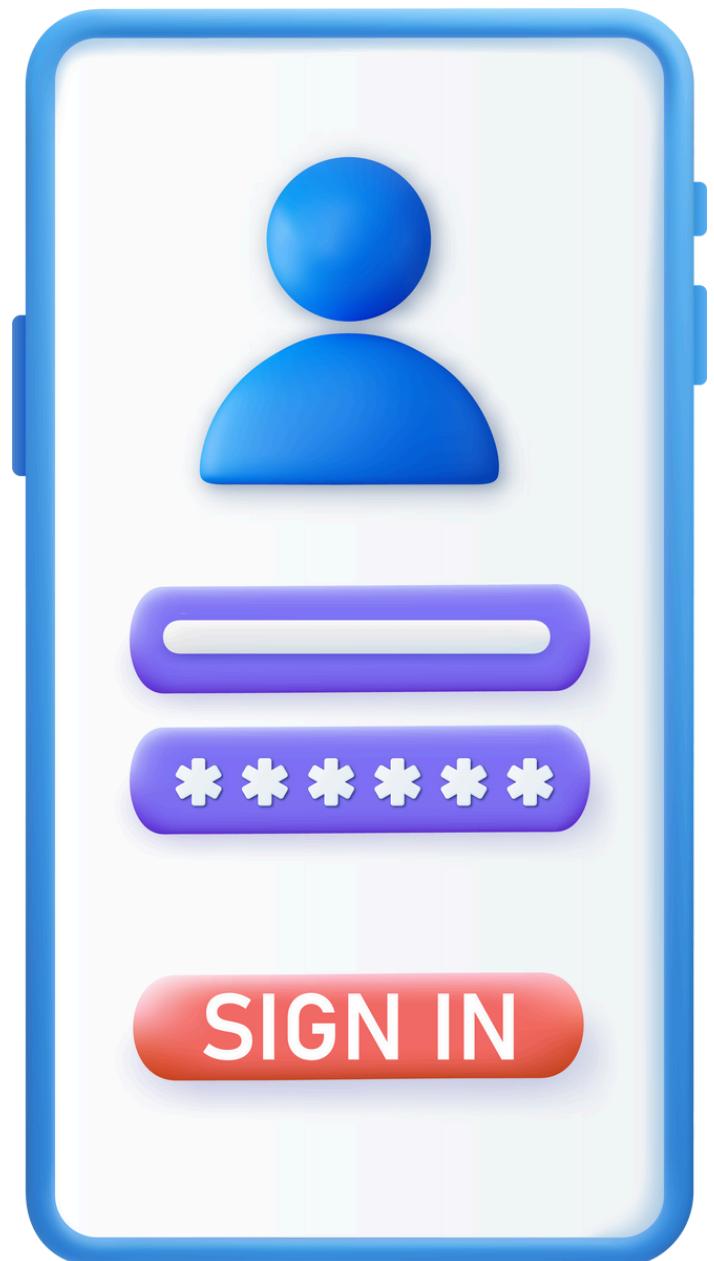




EJERCICIOS

SISTEMA DE CONTROL DE ACCESO

- Mejoremos nuestro sistema de control de acceso.
- Recordando: Nuestro sistema nos solicita al usuario su nombre y contraseña.
- Si existe una coincidencia con los existentes, se le debe permitir la entrada, en caso contrario, se muestra un mensaje de error según corresponda.
- Ahora, cada vez que exista un error, el sistema debe volver a pedir al cliente el ingreso del usuario y la contraseña.
- Es importante aclarar que, si un usuario falla 3 veces, el sistema le va a mostrar un mensaje diciendo que su cuenta se encuentra bloqueada.





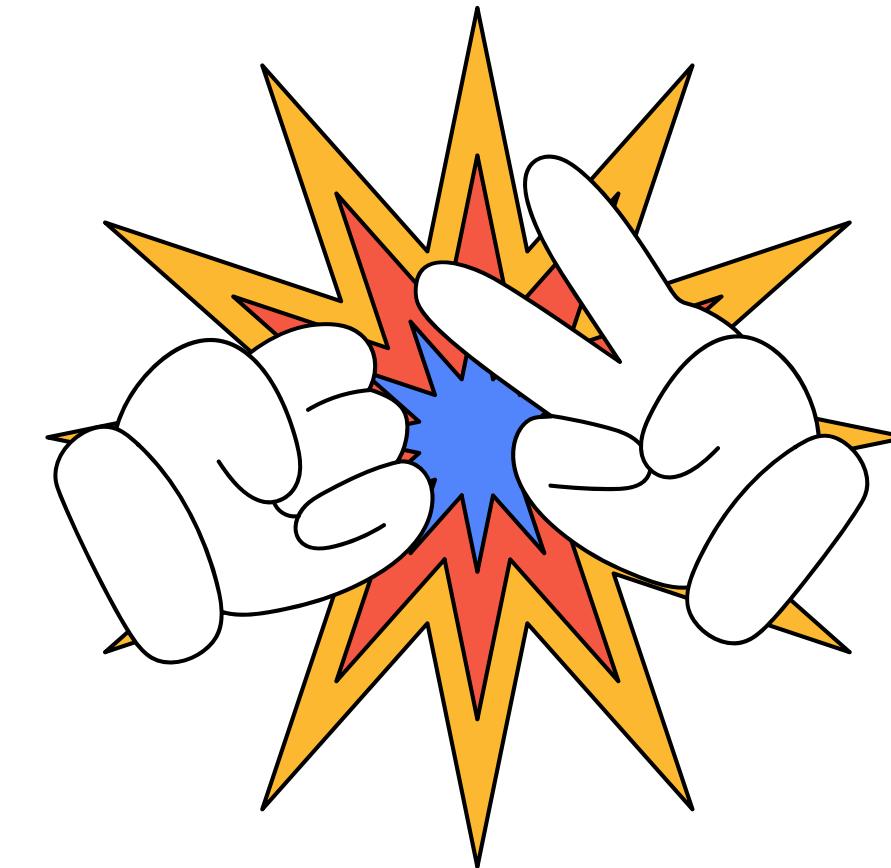
EJERCICIOS

PIEDRA, PAPEL O TIJERA

- Vamos a construir un pequeño videojuego en donde competiremos directamente con la “maquina”.
- Crearemos un menú de opciones para poder seleccionar en cada ronda: Piedra, Papel o Tijera.
- Además, tanto el usuario como la “maquina” van a contar con un contador de 3 vidas. El primero que llegue a 0 pierde el juego.

Tips a tener en cuenta:

- Si utilizamos la función AZAR(n), podremos generar un número aleatorio para que el funcionamiento de la maquina parezca mucho más “real”. **Ejemplo:** AZAR(3), nos puede retornar un número entre 0 y 2.





AWAKELAB

#programmingbootcamp

nodovirtual.awakelab.cl

 jELOU futurejob by  Adalid Chile