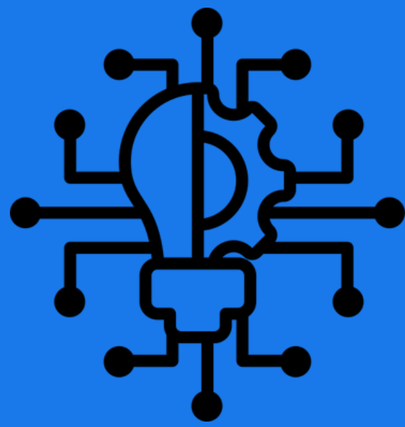
PROGRAMACION AVANZADA

ING. INDUSTRIAL

Esp. Ing.Prof.: Cristina Delia Cruz

FUNCIONESEN C++

Unidad 2



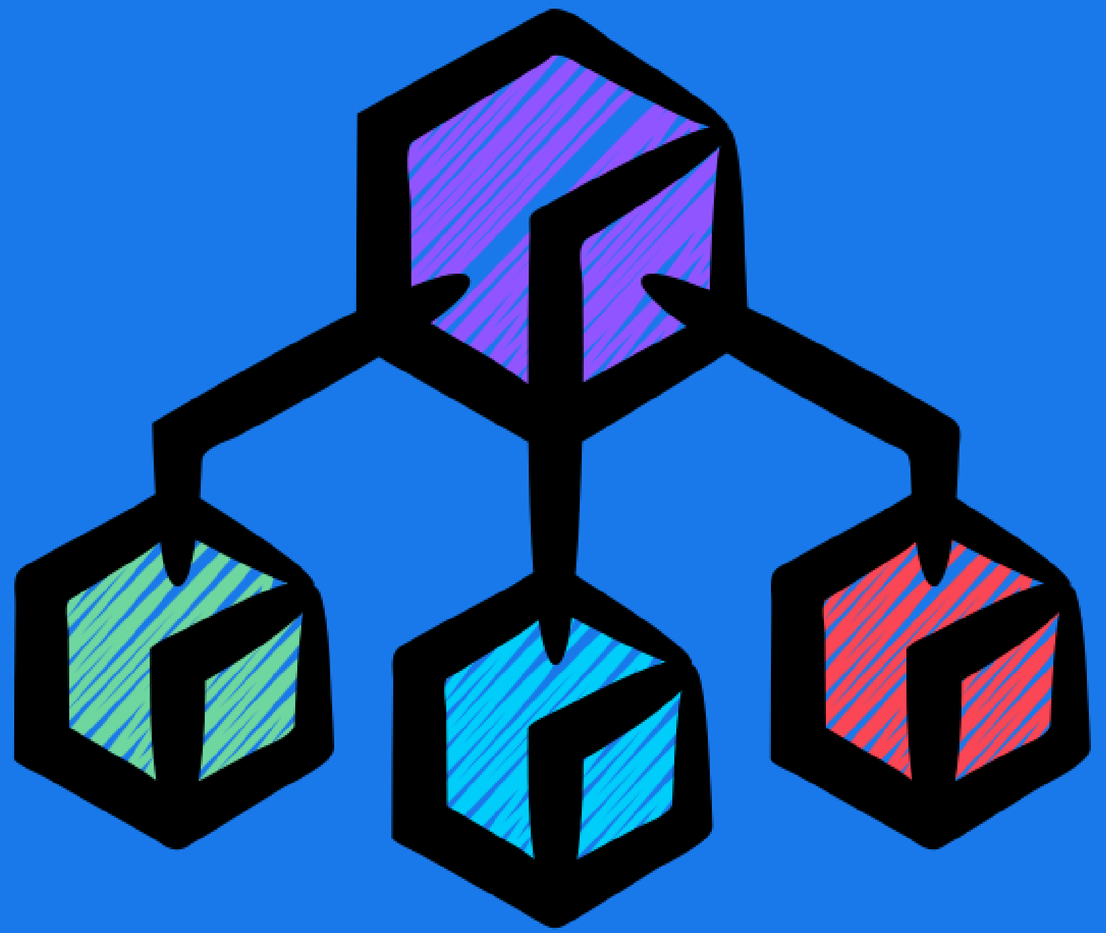
DIVIDE Y VENCERAS

Divide y Vencerás es una estrategia de diseño de algoritmos y resolución de problemas que **consiste en dividir un problema grande en varios subproblemas más pequeños, resolver cada uno de estos subproblemas, y luego combinar las soluciones de estos subproblemas para obtener la solución del problema original.**

DIVIDE Y VENCERAS

Proceso de "Divide y Vencerás":

1. **Dividir**: El problema original se divide en dos o más subproblemas más pequeños, que son instancias más simples del problema inicial.
2. **Resolver (o vencer)**: Cada uno de estos subproblemas se resuelve de manera recursiva (es decir, utilizando el mismo enfoque para resolver problemas más pequeños). Si el subproblema es lo suficientemente pequeño, se resuelve directamente (es el caso base).
3. **Combinar**: Las soluciones de los subproblemas se combinan para obtener la solución del problema original.



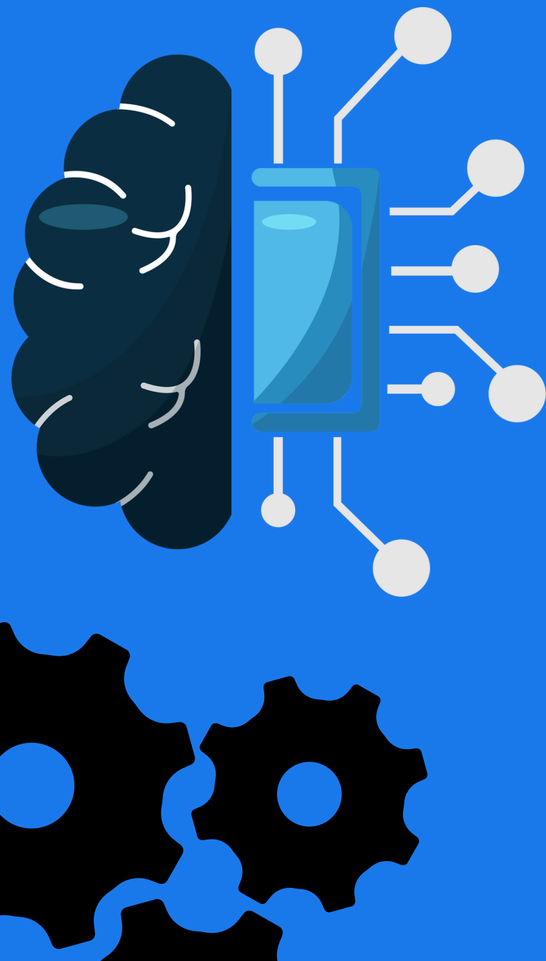
DISEÑO DESCENDETE

El diseño descendente (también conocido como "Top-Down Design") es una metodología de desarrollo de sistemas y software que consiste en descomponer un problema o sistema complejo en partes más pequeñas y manejables. La idea principal es comenzar desde una vista global o abstracta del problema y luego descomponerlo en componentes o subproblemas más específicos y detallados hasta que cada componente pueda ser fácilmente implementado o resuelto.

DISEÑO DESCENDETE

Características del diseño descendente:

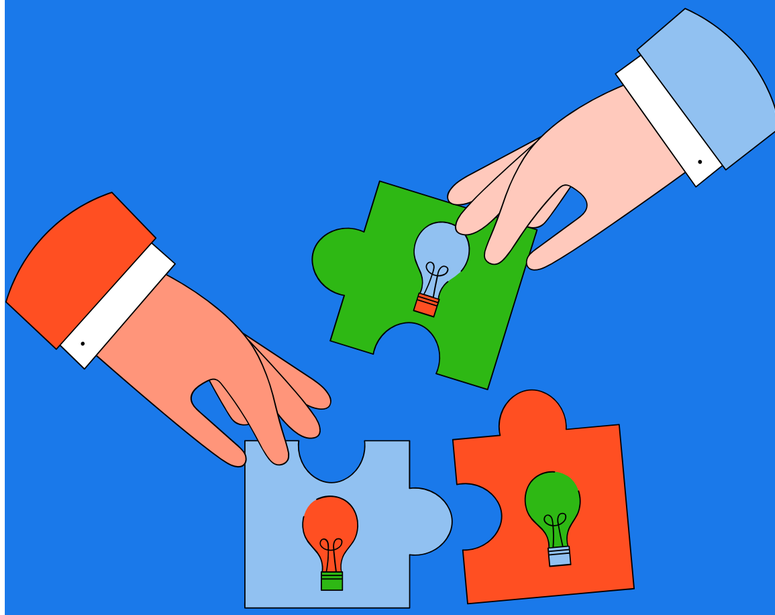
1. Partir del nivel más alto: Se comienza con una descripción general del sistema o problema en su totalidad, sin entrar en detalles específicos.
2. Descomposición en módulos: El problema se descompone en varios subproblemas o módulos que son más sencillos de resolver. Esto continúa en varios niveles hasta que cada submódulo sea lo suficientemente simple para ser resuelto directamente.
3. Implementación paso a paso: Se comienza implementando los niveles más bajos de abstracción. Luego, los módulos pequeños se integran para formar los módulos más grandes, y finalmente el sistema completo.
4. Recursividad en la descomposición: Si uno de los módulos sigue siendo complejo, se repite el proceso de descomposición hasta que se llega a componentes manejables.



DISEÑO DESCENDETE

Ventajas del diseño descendente:

- Organización clara: Facilita una visión clara de todo el sistema y permite que cada parte se desarrolle de manera independiente.
- Facilita la gestión: Como se enfoca primero en los módulos más importantes, permite a los desarrolladores y gestores planificar el proyecto en etapas.
- Modularidad: Como se divide en módulos, facilita el mantenimiento, ya que cada módulo puede ser modificado sin afectar todo el sistema.
- Facilidad en la documentación: Es más fácil documentar el sistema, ya que se construye gradualmente desde los niveles más abstractos hasta los más detallados.



DISEÑO DESCENDETE

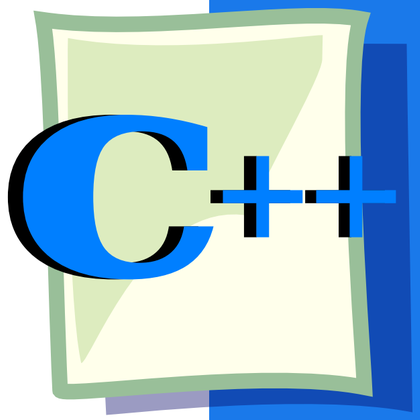
Desventajas:

- Desconexión de los detalles: El enfoque descendente puede a veces ignorar detalles importantes en las primeras etapas, lo que puede causar problemas cuando se llega a niveles inferiores.
- Riesgo de errores en la descomposición: Si el sistema no se descompone correctamente o los módulos no están bien definidos, el diseño puede ser ineficaz.
- Tiempo inicial: Al comenzar con una planificación general, puede requerir más tiempo al principio antes de comenzar la implementación detallada.

DISEÑO DESCENDETE

Ejemplo en C++:

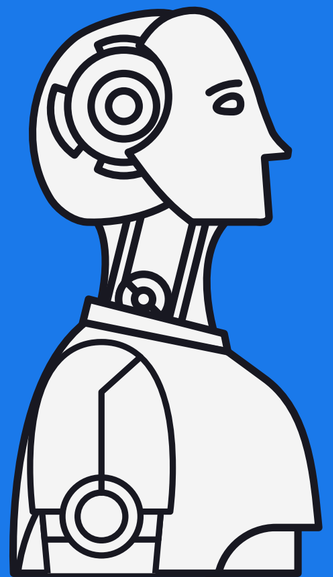
Supongamos que queremos diseñar un sistema de facturación para un supermercado.



1. **Definición del problema general:**
   - El sistema debe manejar productos, facturas y clientes.
2. **División en módulos principales:**
   - **Módulo de productos.**
   - **Módulo de facturas.**
   - **Módulo de clientes.**
3. **Descomposición de un módulo (facturas):**
   - **El módulo de facturas puede tener submódulos para agregar productos a la factura, calcular el total, aplicar descuentos, etc.**
4. **Refinamiento:**
   - El submódulo de "calcular total" puede descomponerse aún más en "sumar precios" y "calcular impuestos".
5. **Implementación:**
   - Comenzamos implementando el cálculo de los precios de los productos, luego sumamos los impuestos y finalmente integramos el módulo de facturación completo.

FUNCIONES

Las funciones son un concepto fundamental en la programación, ya que **permiten organizar el código en bloques reutilizables**, mejorando la claridad, estructura y mantenimiento del programa. Una **función es un bloque de código que realiza una tarea específica y que puede ser "llamada" o "invocada"** desde cualquier parte del programa para ejecutar esa tarea.



FUNCIONES

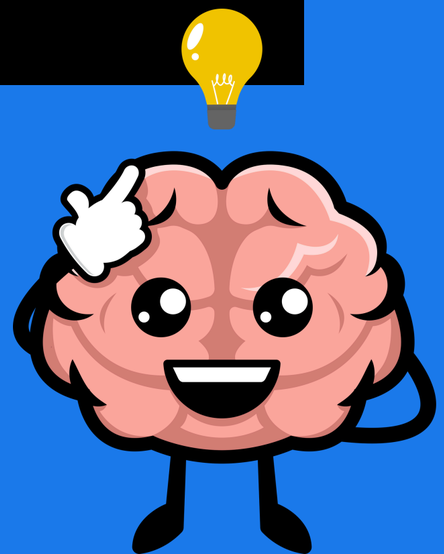
Características principales de una función:

1. **Reutilización del código:** Al definir una función, puedes reutilizar ese bloque de código cada vez que lo necesites sin tener que escribirlo de nuevo. Esto hace que el código sea más eficiente y fácil de mantener.
2. **Modularización:** Dividir un programa en varias funciones facilita la lectura, depuración y actualización del código, ya que cada función tiene un propósito específico.
3. **Parámetros y argumentos:** Las funciones pueden aceptar parámetros, que son valores o variables que se pasan a la función cuando se llama, para que pueda operar con ellos. Los argumentos son los valores que se envían a la función cuando es invocada.
4. **Retorno de valores:** Algunas funciones pueden devolver un resultado o un valor cuando finalizan. Este valor devuelto puede ser utilizado en otras partes del programa.

FUNCIONES

Estructura básica de una función en C++:
Una función en C++ tiene la siguiente estructura general:

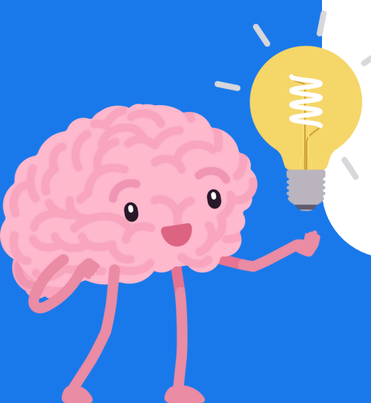
```
tipo_de_retorno nombre_de_función(tipo_parametro1 nombre_parametro1,
tipo_parametro2 nombre_parametro2, ...) {
    // Bloque de código de la función
    return valor;  // opcional, si la función devuelve un valor
}
```

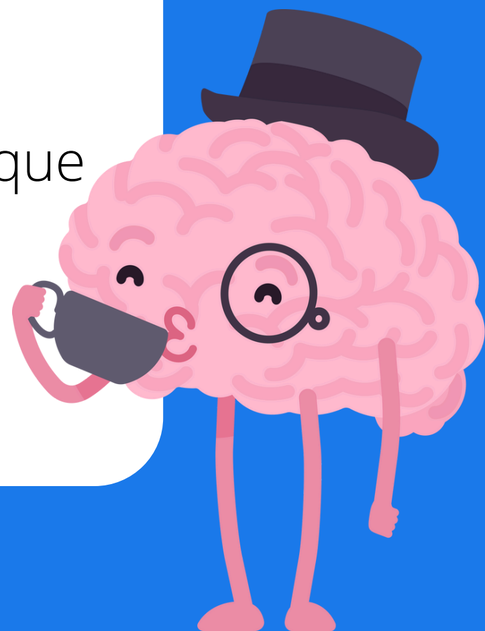


FUNCIONES

Componentes:

1. **Tipo de retorno:** Es el tipo de dato que la función devuelve, como int, double, char, void (si no devuelve nada), etc.
2. **Nombre de la función:** El nombre que se le da a la función para poder invocarla en otras partes del código.
3. **Parámetros (opcional):** Son los valores que la función recibe cuando es llamada. Pueden ser variables o constantes. Una función puede no recibir parámetros, en cuyo caso se deja vacío el paréntesis.
4. **Bloque de código:** Es el conjunto de instrucciones que definen lo que hace la función.
5. **Retorno (opcional):** Si la función devuelve un valor, se utiliza la instrucción return seguida del valor a devolver.





TIPOS DE FUNCIONES

Funciones sin retorno: Son funciones que realizan una acción pero no devuelven ningún valor. Su tipo de retorno es void.

```
void saludar() {
    cout << "Hola, bienvenido!" << endl;
}
```

TIPOS DE FUNCIONES

Funciones con retorno: Son funciones que realizan una acción y devuelven un valor. El tipo de retorno depende del tipo de dato que se devuelve.

```
int sumar(int a, int b) {
    return a + b;
}
```

TIPOS DE FUNCIONES

Funciones con o sin parámetros: Algunas funciones requieren parámetros para trabajar, mientras que otras no los necesitan.

Con Parámetros

```
double multiplicar(double x, double y) {
    return x * y;
}
```

Sin Parámetros

```
void mensaje() {
    cout << "Este es un mensaje sin parámetros." << endl;
}
```

PARÁMETROS

Por valor: **Los parámetros se pasan a la función como copias. Si la función los modifica, esa modificación no afecta a las variables originales en el programa.**

```
void incrementar(int x) {
    x = x + 1;
}
```

En este caso, si pasas una variable a incrementar, la variable original no se verá afectada.

PARÁMETROS

Por referencia: Los parámetros se pasan como referencia, lo que significa que cualquier cambio en los parámetros dentro de la función afectará a las variables originales.

```
void incrementar(int &x) {
    x = x + 1;
}
```

EJEMPLOS

Ejemplo 1. Función sin retorno (void): Sistema de Bienvenida a la Planta.

Crea una función llamada darBienvenida que no reciba parámetros ni devuelva valor. Su propósito es imprimir un mensaje de bienvenida a los operadores al inicio de un turno de producción. Desde la función principal, llama a esta función para dar la bienvenida.

EJEMPLOS

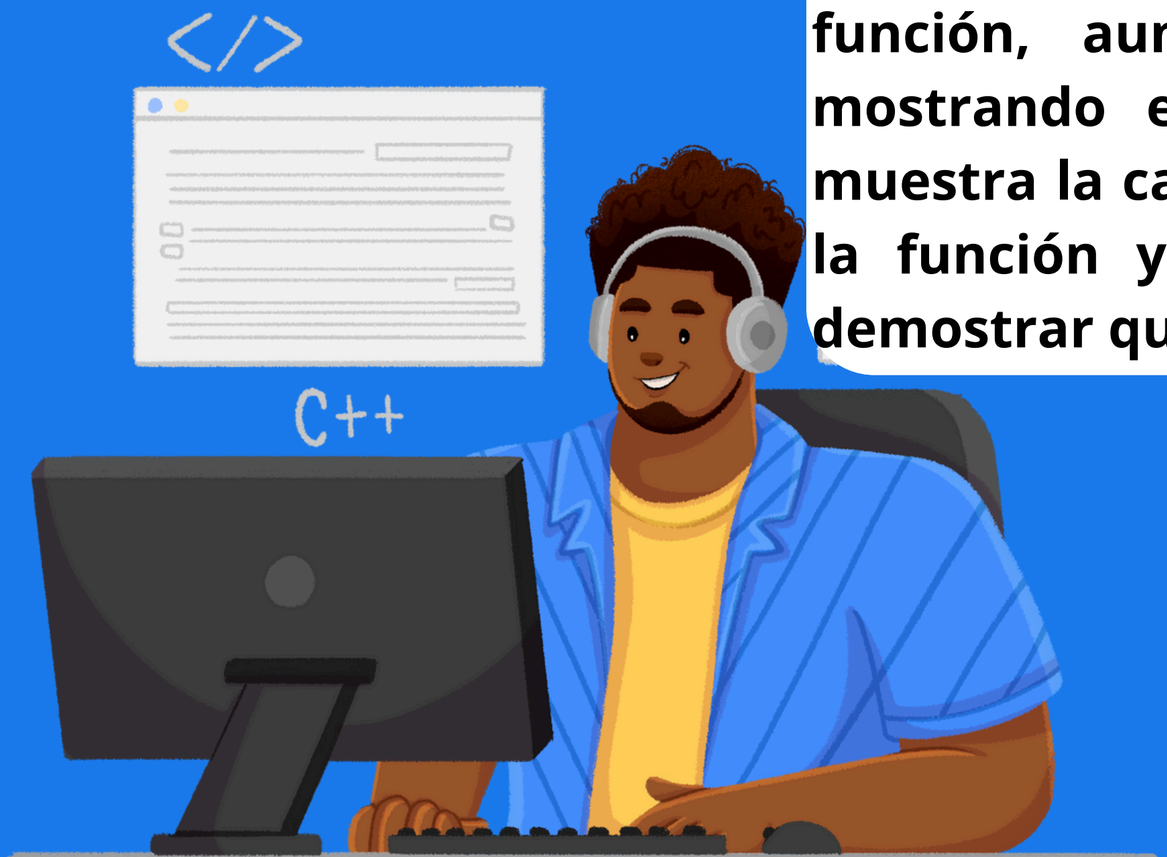
Ejemplo 2. Función con retorno y parámetros: Cálculo de Costo Total de Producción

Escribe una función llamada calcularCostoTotal que reciba el costo unitario y la cantidad de unidades producidas. La función debe multiplicar ambos valores y devolver el costo total. En el programa principal, pide estos valores al usuario y muestra el resultado.

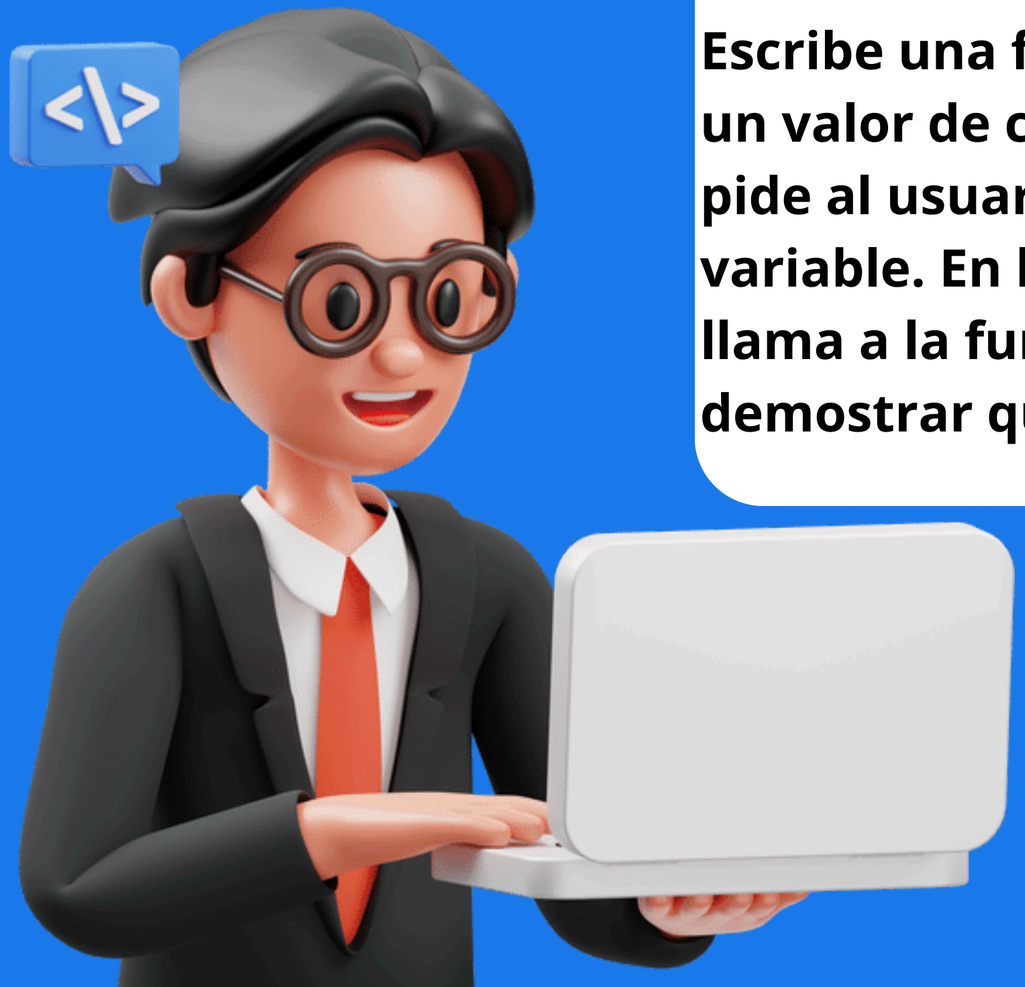
EJEMPLOS

Ejemplo 3. Parámetros por valor: Simulación de un Incremento de Inventario.

Crea una función llamada simularRecepcion que reciba la cantidad de inventario actual por valor. Dentro de la función, aumenta esta cantidad en 50 unidades, mostrando el nuevo valor. En la función principal, muestra la cantidad inicial antes de la llamada, llama a la función y muestra la cantidad nuevamente para demostrar que el valor original no cambia.



EJEMPLOS



Ejemplo 4. Parámetros por referencia: Ajuste de la Calibración de una Máquina
Escribe una función llamada ajustarCalibracion que reciba un valor de calibración por referencia. Dentro de la función, pide al usuario un nuevo valor de calibración y lo asigna a la variable. En la función principal, muestra el valor original, llama a la función y luego muestra el valor final para demostrar que la calibración se ajustó permanentemente.

EJEMPLOS

Ejemplo 5. Función de División Segura: Cálculo de Eficiencia de Producción
Consigna: Desarrolla una función llamada calcularEficiencia que reciba la cantidad de productos terminados y la cantidad de materia prima utilizada. La función debe retornar el índice de eficiencia (productos / materia prima). Debe manejar el caso de una posible división por cero, mostrando un mensaje de error y devolviendo 0 si se ingresa 0 en la materia prima.

EJEMPLOS

Ejemplo 6. Función con bucle for: Costo de Horas de Mantenimiento
Crea una función llamada costoMantenimiento que reciba el número de horas de mantenimiento planeadas y el costo por hora. La función debe usar un bucle for para sumar el costo por cada hora y devolver el costo total del mantenimiento.

EJEMPLOS

7. Función con bucle while: Monitoreo de Temperatura de Proceso
Escribe una función llamada monitorearTemperatura que pida al usuario que ingrese la temperatura de un proceso. La función debe usar un bucle while para seguir pidiendo la temperatura hasta que el valor sea menor a 200 grados C, que es la temperatura de seguridad.

EJEMPLOS

Ejemplo 8. Función para el Cálculo del Rendimiento de un Proceso

Consigna: Crea un programa con una función llamada calcularRendimiento que reciba como parámetros la cantidad de materia prima y la cantidad de producto final. La función debe calcular el rendimiento porcentual del proceso ((producto_final / materia_prima) * 100) y devolver el resultado.

En la función principal, pide al usuario que ingrese la cantidad de materia prima utilizada y la cantidad de producto final obtenido. Luego, llama a calcularRendimiento y muestra el rendimiento del proceso.


¿PREGUNTAS?


MUCHAS
GRACIAS!!!
Nos Vemos en la Siguiente Clase