FUNCIONES, BUCLES Y CONDICIONALES

1. Sistema de Aterrizaje Asistido (Aterrizaje en Múltiples Pistas)

Crea un programa para una nave espacial que se aproxima a un planeta con **múltiples pistas de aterrizaje**. El sistema debe pedir al usuario que introduzca el **número de pista** (`int`). Si el número está entre 1 y 5, el sistema debe informar que el aterrizaje es seguro. Si el número es 0, la operación debe cancelarse. Para cualquier otro valor, debe indicar que la pista no es válida y seguir pidiendo un número de pista hasta que se introduzca un valor válido o se cancele la operación. Usa un bucle `do-while` y una función que valide el número de pista.

2. Control de Nivel de Oxígeno

Desarrolla una función llamada `comprobarNivelOxigeno` que reciba un **porcentaje de oxígeno** (`int`). La función debe imprimir "Nivel óptimo" si el porcentaje está entre 80% y 100%, "Nivel bajo, se requiere reabastecimiento" si está entre 40% y 79%, y "Nivel crítico, evacuación de la cabina" si es inferior a 40%. En el programa principal, pide al usuario que ingrese el nivel de oxígeno y llama a la función para evaluar el estado. Repite el proceso con un bucle `for` 3 veces para simular múltiples lecturas.

3. Simulación de Recolección de Meteoritos

Imagina que una **nave de recolección de meteoritos** debe recoger 10 meteoritos. Crea una función `recolectarMeteoritos` que tome como argumento el **número de meteoritos a recolectar** (`int`). Dentro de la función, usa un bucle `for` para simular la recolección, imprimiendo un mensaje por cada meteorito recogido y mostrando el **progreso total** (ej: "Meteorito 1 de 10 recogido. Progreso: 10%"). Al finalizar el bucle, imprime "Misión de recolección completada".

4. Búsqueda de Satélites Activos

Escribe una función `buscarSatelites` que reciba un **código de satélite** (`String`). Dentro de la función, usa una estructura `switch` para determinar el estado del satélite. Si el código es "SAT-A", "SAT-B" o "SAT-C", imprime "Satélite activo, transmitiendo datos". Si el código es "SAT-D" o "SAT-E", imprime "Satélite inactivo, requiere mantenimiento". Para cualquier otro código, informa "Código de satélite desconocido". En el programa principal, usa un bucle `while` para permitir que el usuario consulte el estado de varios satélites hasta que escriba "FIN".

5. Cálculo de Órbita Estabilizada

Crea un programa que ayude a estabilizar la órbita de un satélite. El sistema debe solicitar la **altura de la órbita** (`int`) hasta que se ingrese un valor entre 200 km y 500 km. Usa un bucle `while` para seguir pidiendo la altura hasta que se cumpla la condición. Cuando se ingrese un valor válido, la función `estabilizarOrbita` debe calcular el tiempo de estabilización usando la fórmula `tiempo = (altura - 200) * 1.5` y mostrar el resultado.

---

6. Sistema de Comunicación Interplanetaria

Diseña una función `enviarMensaje` que reciba un **destino** (`String`) y un **mensaje** (`String`). El programa debe pedir al usuario que ingrese el destino (ej: "Marte", "Júpiter") y el mensaje. Dentro de la función, usa una estructura `if-else if-else` para simular la transmisión. Si el destino es "Marte", imprime "Mensaje a Marte enviado, tiempo de llegada: 3 minutos". Si es "Júpiter", imprime "Mensaje a Júpiter enviado, tiempo de llegada: 25 minutos". Para cualquier otro destino, indica "Destino fuera de alcance". El programa debe permitir el envío de **múltiples mensajes** en un bucle `for`.

7. Control de Consumo de Combustible

Desarrolla una función `calcularConsumo` que acepte los **kilómetros recorridos** (`double`) y el **consumo por km** (`double`). La función debe calcular el consumo total y compararlo con un **tanque inicial de 500 litros**. Si el consumo total es menor o igual a 500, imprime "Viaje completado. Combustible restante: X litros". Si el consumo excede los 500 litros, imprime "Advertencia: Combustible insuficiente, se requiere recarga". Usa un bucle `do-while` para pedir los datos al usuario hasta que el viaje sea sostenible (consumo <= 500).

8. Simulación de Navegación por Puntos de Control

Crea un programa que simule la navegación de una nave a través de **5 puntos de control**. Define una función `navegarPuntos` que acepte un **punto de control actual** (`int`). Dentro de la función, usa un bucle `for` para iterar desde el punto 1 hasta el punto 5, imprimiendo un mensaje de "Navegando hacia el punto de control [número]". Si el número del punto de control es par, imprime "¡Control de trayectoria exitoso!". Después de los 5 puntos, imprime "Ruta completada".

---

9. Generador de Códigos de Misión

Escribe un programa con una función `generarCodigoMision` que reciba el **nombre de una misión** (`String`). La función debe crear un **código de misión** que sea una combinación del nombre de la misión y un número secuencial aleatorio del 1 al 100. En el programa principal, usa un bucle `while` para permitir al usuario generar **códigos para múltiples misiones** hasta que decida salir. La función debe mostrar el código generado. Si el nombre de la misión está vacío, debe imprimir "Nombre de misión inválido".

10. Reporte de Despliegue de Paneles Solares

Desarrolla una función `desplegarPaneles` que tome como argumento el **número de paneles a desplegar** (`int`). La función debe usar un bucle `for` para simular el despliegue de cada panel, imprimiendo "Desplegando panel [número]". Si el número del panel es un múltiplo de 3, debe imprimir "Panel [número] desplegado con éxito, calibración automática completada". Al final del bucle, si todos los paneles se desplegaron, imprime "Despliegue de paneles solares completado. Energía al 100%". Si no se desplegó ningún panel, imprime "No se desplegó ningún panel". Pide al usuario el número de paneles y llama a la función.