1) **El símbolo '=':**
   a. **¿Qué tipo de operador es?**

      Es un operador de asignación.
      Puede formar parte de un operador relacional (aunque por sí mismo no lo es).

   b. **¿En qué lugares de un programa puede aparecer?**

      - En la declaración e inicialización de una variable.
      - En una sentencia de asignación.
      - Como parte de un operador relacional (`!=`, `<=`, `>=` o `==`), en una expresión relacional o lógica.

   c. **¿Cuál es su significado?**

      Asigna a la variable situada a su izquierda el valor de la expresión situada a su derecha.

2) **Si una variable no ha sido inicializada y es incluida en una instrucción de asignación, ¿qué situaciones pueden darse? ¿qué tipo de error podría cometerse? Razona las respuestas.**

   Si la variable aparece en la parte izquierda de la instrucción de asignación, ésta tomará como valor inicial el que resulte de la expresión situada en la parte derecha de la misma.

   Si la variable aparece en la parte derecha de la instrucción de asignación formando parte de una expresión, el resultado de ésta será indeterminado y la ejecución de la sentencia provocará un efecto no previsto ya que su valor puede ser cualquiera (lo que en ese momento almacene la zona de memoria que el traductor le haya asignado).

   Por ejemplo, si la variable `n` no está inicializada y el programa realiza la instrucción `ESCRIBIR n + 7`, la salida por pantalla corresponderá a un valor aleatorio.

3) **Un programa debe realizar una *tarea1* si se da una *condición1* y debe realizar una *tarea2* si se da una *condición2*. ¿Cómo se debería programar si puede que se tengan que realizar alguna de esas tareas o que se tengan que realizar las dos o que no se tenga que realizar ninguna?**

   Implementando estructuras condicionales no anidadas:

```
SI condicion1
  <tarea1>
FIN SI

SI condicion2
  <tarea2>
FIN SI
```

   De esta manera se evalúan siempre las dos condiciones y se realizan las tareas que deban ejecutarse en función del cumplimiento o no de éstas (pudiendo ejecutarse todas, ninguna o alguna de ellas).

4) **¿Por qué no es conveniente utilizar una estructura `SEGÚN_VALOR` si la expresión que se va a evaluar es de tipo "*real*"?**

Porque un valor real corresponde con una cantidad "no discreta", es decir, con muchos más valores posibles (fraccionarios) que si fuera entera ("discreta") y muy probablemente no se puedan definir suficientes tratamientos individuales ("*case*").

5) **Sobre los bucles `MIENTRAS` y `REPETIR`:**
   a. **¿Cuáles son sus diferencias?**

- En MIENTRAS la evaluación de la condición de salida se ejecuta antes que las instrucciones del bucle. En REPETIR se ejecuta después.
- Las instrucciones del bucle REPETIR se ejecutan al menos una vez, las del bucle MIENTRAS pueden no ejecutarse nunca.

   b. **¿Cómo validar un dato utilizando una estructura `MIENTRAS` en lugar de una estructura `REPETIR`? ¿Tiene alguna desventaja?**

```
ESCRIBIR "Introducir número"
LEER numero

MIENTRAS numero < 0
  ESCRIBIR "Introducir número"
  LEER numero
FIN MIENTRAS
```

La desventaja que tiene este tipo de estructura es que las instrucciones de entrada/salida hay que escribirlas dos veces en el código fuente, una para la primera ejecución y otra para la repetición (en caso de que el dato a validar no sea correcto).

6) **Sobre el bucle `PARA`:**
   a. **¿En qué circunstancias no es posible/aconsejable utilizar un bucle `PARA`?**

En general, el bucle PARA puede utilizarse siempre. Las situaciones descritas a continuación pueden (o no) hacerlo desaconsejable:

- Cuando no se conozca el nº exacto de iteraciones que el bucle debe ejecutar.
- Cuando pueda darse el caso de tener que salir del bucle antes de que el contador alcance el valor máximo.
- Cuando no se quieran utilizar instrucciones de salto incondicional (break, goto).

   b. **¿Hay que declarar el contador? ¿hay que inicializarlo? ¿puede utilizarse en una instrucción de asignación?**

- Hay que declararlo como cualquier otra variable.
- El propio bucle se encarga de inicializarlo, no hace falta hacerlo previa y explícitamente.
- Puede utilizarse en la parte derecha de una instrucción de asignación, es decir, como operando de una expresión como cualquier otra variable.

- Puede utilizarse en la parte izquierda de una instrucción de asignación aunque eso provocará que su valor va a cambiar y requerirá que se controle esa situación (sobre todo en la condición de salida). En este caso no es recomendable y suele emplearse alguna otra estrategia.

7) Describe con un ejemplo el uso de una variable de tipo "*bandera*" en un programa.

Permiten dirigir la ejecución de las sentencias por "caminos" distintos en función de su valor (lógico) de una manera sencilla y sin realizar ningún tipo de cálculo.

Pueden utilizarse en cualquier estructura de control (selectiva o repetitiva).

Un ejemplo podría ser éste: utilizamos el interruptor `POSITIVO` para registrar si un número es mayor que cero. Después, y dependiendo de su valor, el programa realiza tareas distintas:

```
SI numero > 0
  positivo = VERDADERO
SI NO
  positivo = FALSO
FIN SI

SI positivo == VERDADERO
  <Acciones>
  . . . . . .
SI NO
  <Acciones>
  . . . . . .
FIN SI
```

8) ¿Para qué sirve documentar el código fuente de un programa, qué utilidad puede tener y en qué situaciones?

Permite entender las tareas que realiza el programa de una manera fácil al estar explicadas en lenguaje natural, sin tener que hacer previamente un examen exhaustivo a las instrucciones escritas en un lenguaje de programación.

Se facilita así la labor del programador que deba acceder al código fuente para alguna tarea relacionada con el mantenimiento (búsqueda de errores, añadido de funcionalidades, etc.).