BUCLES WHILE, FLAG Y FOR EN R

Bucle While

Uso: se utiliza en situaciones donde necesitas repetir una acción de manera indefinida hasta que se cumpla una condición específica. Cuando no se sabe cuántas veces se ejecutará el bucle.

- **Casos:**
  - **Validar entrada del usuario:**
    Cuando se necesita que el usuario introduzca una entrada válida: pedir un acontraseña
  - **Evitar bucle infinito:**
    Asegurarse de que la condición se vuelva falsa en algún punto o se puede usar break para salir del bucle.

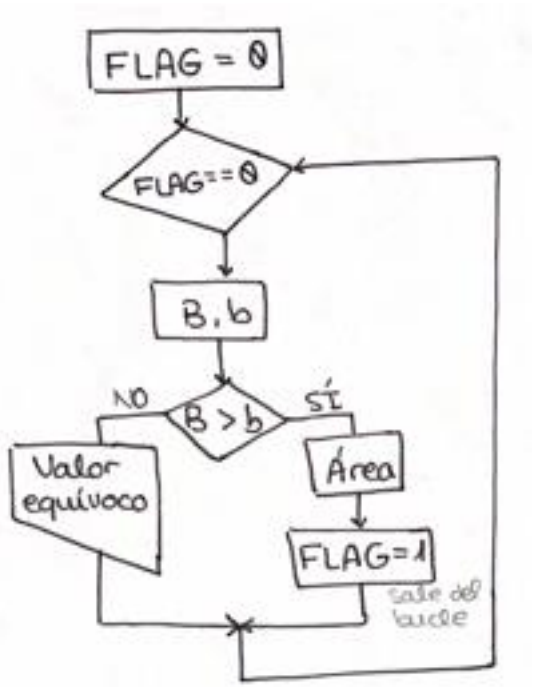
Ejemplo:

IMPORTANTE: a la hora de programar 1º fijarse en lo que hay a la derecha del igual, la fórmula u operación que se quiera calcular, y lo valores que aparezcan en ella tienen que haber sido obtenidos previamente.

- Hasta que…: poner en while lo contrario a lo que queremos que se cumpla
- Mientras que…: ponemos en while la condición

```
#Ingresar los valores necesarios: Base mayor, base menor y altura
    a=as.numeric(readline(prompt="Ingresar valor de B:"))
    b=as.numeric(readline(prompt="Ingresar valor de b:"))
    c=as.numeric(readline(prompt="Ingresar valor de h:"))
#Hasta que no se cumpla una condición: poner error, luego pedir datos, calcular el área del trapecio en este caso, si los datos que el usuario ha introducido son los correctos.
    while (a<=b){
    print(paste("Incoherencia, vuelve a meter los datos"))
    a=as.numeric(readline(prompt="Ingresar valor de B:"))
    b=as.numeric(readline(prompt="Ingresar valor de b:"))
    c=as.numeric(readline(prompt="Ingresar valor de h:"))
    d=(a+b)*c/2
#Mostrar el valor del área del trapecio
    print(paste("El valor del área del trapecio es:", d))
    }
```

Bucle Flag



Uso: para controlar la ejecución y salida de un bucle. Cuando se quiere detener un bucle bajo condiciones específicas que son más difíciles de manejar con while.

1.**Variable bandera (flag)**: se inicializa antes de entrar en el bucle.
2.**Condición dentro del bucle**: se evalúa la variable bandera en cada iteración. Si la condición se cumple, se actualiza la bandera para finalizar el bucle o modificar su comportamiento.

- **Casos:**
  - Solicitar entrada hasta cumplir una condición
  - Bucle infinito con salida controlada

Ventaja:

- **Control flexible**: permite controlar el flujo del bucle de manera clara y modular.
- **Condiciones múltiples**: se puede combinar con varias condiciones en lugar de depender de la única expresión de while.
- **Legibilidad**: puede mejorarla cuando el criterio de salida del bucle es complejo.

Consideraciones

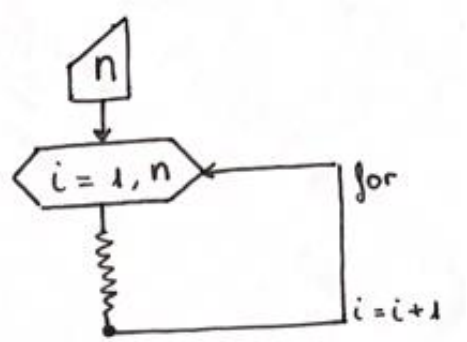
- El bucle debe estar bien diseñado para evitar ciclos infinitos. Asegúrate de que la condición de la bandera pueda cambiar durante la ejecución.
- Utiliza mensajes de depuración (cat() o print()) para rastrear el estado de la bandera en bucles más complejos.

Progama:

```
	FLAG=0 #Variable de control para el bucle
#Ingresar los valores necesarios: Base mayor, base menor y altura
	while(FLAG==0){
	a=as.numeric(readline(prompt="Ingresar valor de B:"))
	b=as.numeric(readline(prompt="Ingresar valor de b:"))
	c=as.numeric(readline(prompt="Ingresar valor de h:"))
	print(paste("Incoherencia, vuelve a meter los datos"))
		if(a>b){
	d=(a+b)*c/2
	print(paste("El valor del área del trapecio es:", d))
	FLAG=1
		}
	}
```

Bucle For

Uso: para iterar sobre secuencias de elementos, como vectores, listas, rangos. Es útil para ejecutar un bloque de código repetidamente mientras se recorre cada elemento de una colección.



En este bucle sabemos cuántas veces se va a repetir, o bien porque lo indiquemos directamente en el programa o porque lo pedimos como valor de entrada.

Ventaja:

- **Simplicidad y claridad**:
  - La sintaxis del bucle for es fácil de entender y sigue un flujo lógico.
  - Ideal para principiantes en R, ya que es más explícito que funciones como apply o lapply.
- **Control detallado sobre la iteración**:
  - Permite realizar operaciones personalizadas en cada iteración.
  - Puedes usar break para salir del bucle o next para saltar a la siguiente iteración.
- **Versatilidad**:
  - Funciona con diferentes estructuras de datos (vectores, listas, matrices, data frames).
  - Útil para manipular elementos de forma individual, especialmente cuando necesitas realizar varias operaciones en cada elemento.
- **Adecuado para procesos iterativos dependientes**:
  - Ideal cuando el cálculo en una iteración depende de los resultados de las iteraciones previas, como en algoritmos recursivos o simulaciones.

Casos de uso:

- **Tareas iterativas simples**: cuando necesitas recorrer una estructura para aplicar una operación básica.

Ejemplo:

```
numeros <- c(1, 2, 3, 4, 5)
for (n in numeros) {
  print(n * 2)
}
```

- **Procesos dependientes de iteraciones previas**:

Ejemplo: Calcular una serie acumulativa.

```
resultado <- 0
for (i in 1:10) {
  resultado <- resultado + i
  print(resultado)
```

}

- **Manipulación de estructuras complejas**: si trabajas con listas anidadas o estructuras que requieren procesamiento iterativo paso a paso.

Ejemplo: Modificar valores en una lista dependiendo de una condición.

- **Condiciones avanzadas dentro del bucle**: cuando necesitas lógica condicional para decidir qué hacer en cada iteración.

Ejemplo:

```
for (i in 1:10) {
  if (i %% 2 == 0) {
    print(paste(i, "es par"))
  } else {
    print(paste(i, "es impar"))
  }
}
```

Consideraciones:

- **Desempeño**: son más lentos que las operaciones vectorizadas debido a la naturaleza interpretada de R.
  - Alternativas como apply, lapply, o directamente usar funciones vectorizadas: la función operará sobre todos los elementos de un vector sin necesidad de recorrerlos y actuar sobre cada elemento de a uno por vez.
- **Evitar bucles innecesarios**: muchos problemas pueden resolverse mediante funciones vectorizadas.

Ejemplo vectorizado:

```
# Con bucle
        numeros <- c(1, 2, 3, 4, 5)
        dobles <- c()
        for (n in numeros) {
          dobles <- c(dobles, n * 2)
        }
# Vectorizado
        dobles <- numeros * 2
```

- **Legibilidad con iteraciones anidadas**: los bucles anidados pueden complicar el código y hacerlo más difícil de depurar o escalar.

Recomendaciones:

- Usa for cuando el control detallado y la claridad del flujo sean esenciales.
- Considera alternativas vectorizadas (apply, sapply, mapply) para mejorar el rendimiento.

- Si trabajas con grandes cantidades de datos, evalúa el uso de librerías optimizadas como data.table o paralelización (foreach, parallel).
- Optimiza el uso del bucle preasignando estructuras (como vectores o matrices) para evitar el crecimiento dinámico de objetos dentro del bucle.

Estructura básica:

```
for (variable in secuencia) {
        #Código a ejecutar en cada interacción
}
```

Progama:

```
#Calcular la raiz cuadrada de un número S
n=as.numeric(readline(prompt="ingresar el valor de n, número de interacciones: "))
S=as.numeric(readline(prompt="ingresar el valor de S, valor de la raiz cuadrada: "))
x=S/2 #x tiene que tener un valor inicial entonces pones la fórmula como si no hubiese x
for(i in 1:n){
        x=1/2*(x+S/x) #1º pones la fórmula que es lo que te dan, luego miras los datos que necesitas saber antes de hacer la fórmula.
}
print(x) #valor aproximado
vex=sqrt(S) #valor real
print(paste("el error cometido es", abs(x-vex))) #abs es valor absoluto
```