Bucles, guía práctica: Cómo hacer programas o fragmentos de programa para resolver un problema con bucles.

Hay muchas maneras de realizar correctamente un programa empleando bucles, las transformaciones de programas permiten pasar de unos a otros siendo todos ellos soluciones perfectamente válidas. En esta guía mostraremos un mecanismo sistemático para obtener de forma simple y metódica soluciones a los problemas presentados.

Primero, hay que emplear bucles cuando debe *repetirse* algo. Si no hay necesidad de repetir alguna cosa los bucles dejan de ser necesarios. Segundo, hay otras alternativas a bucles como pueda ser la recursión, que quedan fuera del ámbito de la materia. Y tercero como ya se mencionó al principio, ésta es solamente una de las muchas formas que hay para resolver estos tipos de problemas; pueden emplearse otras y llegar a soluciones diferentes que también sean válidas.

Pasos a seguir.

Son seis, los describiremos brevemente y luego mostraremos varios ejemplos de uso que serán la mejor forma de explicar el método.

1) Determinar si es un bucle *while* o *for*. Si el número de repeticiones es determinado será un bucle *for*, de lo contrario será un bucle *while*. Ejemplo: sumar los números pares entre dos números dados como dato, es un bucle *for*. Mejorar la precisión de una raíz mientras el error sea mayor que cierto valor, es un bucle *while*.

2) Diseñar en primer lugar lo que se repite dentro del bucle con las variables que hagan falta y comprobar que se pasa correctamente de una iteración a la siguiente. Es conveniente emplear para esto unos valores de prueba y lápiz en ristre hacer una tabla con la variables relevantes comprobando que se modifican de acuerdo a lo previsto y que además los cambios se hacen de tal forma que el bucle pueda finalizar en algún momento.

3) Determinar la condición de ejecución si es un bucle *while* o el rango si es un *for*. A veces es más fácil encontrar la condición de terminación, basta recordar que la condición de ejecución es la negada de la de terminación.

4) Determinar y poner valores iniciales antes de entrar en el bucle. Casi siempre hay que inicializar valores para las variables que intervienen en un bucle, debemos asegurar que dentro del bucles las variables que intervienen están todas definidas y que tienen los valores iniciales adecuados. En el caso del bucle *while* esto incluye la condición de ejecución. Conviene comprobar que la primera vez que se ejecute el bucle todo toma el valor correcto.

5) Finalizado el bucle obtener lo pedido a partir de los últimos valores de las variables cuando se ejecutó por última vez el bucle. Es decir, extraer el resultado a partir del estado en que quedaron las variables cuando el programa salió del bucle. No hay que olvidar además que los bucles pueden no ejecutarse, en el caso del *while* si la condición de ejecución es ya inicialmente falsa y en el caso del *for* si el valor inicial es mayor o igual que el final en los bucles crecientes y menor o igual en los decrecientes. Estos casos donde el cuerpo del bucle puede no ejecutarse deben también ser tenidos en cuenta.

6) Si fuera necesario, añadir un condicional dentro del bucle si se requiere salir para procesar el último elemento de forma distinta de los demás (y hacerlo ya fuera del bucle).

Ejemplo 1: Hacer un programa que calcule la suma de todos los números que se proporcionen por teclado. El programa finalizará cuando se introduzca un cero.

1) No sabemos cuántos números se van a introducir, el problema se resolverá entonces con un bucle *while*. Nuestro programa tiene de momento este aspecto:

```
???
while ???:
    ???
    ???
???
```

2) ¿Qué es lo que hay que repetir? Pues pedir un número por teclado y añadirlo a una suma. El programa es ahora:

```
???
while ???:
    numero=float(input("número:"))
    suma=suma+numero
???
```

3) ¿Cuándo finaliza el bucle? Cuando el número sea un cero, por tanto debe ejecutarse mientras el número no sea un cero. Nuestro programa es:

```
???
while numero!=0:
    numero=float(input("número:"))
    suma=suma+numero
???
```

4) ¿Qué variables intervienen en el bucle y qué valores deben tener? Intervienen *numero* (en la expresión booleana) y *suma*. Es claro que si vamos a acumular valores en *suma*, esta variable debe ser inicialmente cero. En cuanto a *numero* debemos asignarle un valor inicial para asegurar que se entra correctamente en el bucle y además que este valor no interfiere con los cálculos. Cualquier valor distinto de cero es correcto, se entra en el bucle e inmediatamente se pide un nuevo valor para esta variable de manera que se elija el que se elija –distinto de cero- no afectará al correcto funcionamiento del programa. El programa tiene ahora esta pinta:

```
suma=0.0
numero=1.0        # sirve cualquiera distinto de cero
while numero!=0:
    numero=float(input("número:"))
    suma=suma+numero
???
```

5) ¿Qué resultado se pide? En este caso la suma, que ya la tenemos, no hay que realizar ninguna operación más, se puede mostrar ya por la salida. El programa queda:

```
suma=0.0
numero=1.0        # sirve cualquiera distinto de cero
while numero!=0:
    numero=float(input("número:"))
    suma=suma+numero
print("la suma es",suma)
```

6) ¿Hay dentro del bucle algún caso que nos obligue a salir de él? Realmente el bucle debería finalizar en cuanto se introduce un cero, pero sumar cero a lo que sea no va a afectar el resultado de manera que podemos dar el programa por terminado.

Ejemplo 2: Hacer un programa que calcule el promedio de todos los números que se proporcionen por teclado. El programa finalizará cuando se introduzca un número negativo.

1) No sabemos el número de veces luego es un bucle *while*:

```
???
while ???:
    ???
    ???
???
```

2) Necesitamos la suma y la cantidad de números introducidos. Usaremos un contador.

```
???
while ???:
    numero=float(input("número:"))
    contador=contador+1
    suma=suma+numero
???
```

3) La condición de terminación es que el número sea negativo, luego el bucle debe repetirse mientras el número sea mayor o igual a cero:

```
???
while numero>=0:
    numero=float(input("número:"))
    contador=contador+1
    suma=suma+numero
???
```

4) Hay que dar valores iniciales a *suma*, *contador* y *numero*: Los dos primeros van a ser cero, uno real y otro entero. Para *numero* vale cualquier valor mayor o igual que cero:

```
contador=0
suma=0.0
numero=1.0        # sirve cualquiera mayor o igual a cero
while numero>=0:
    numero=float(input("número:"))
    contador=contador+1
    suma=suma+numero
???
```

5) Se pide el promedio, el cual se obtiene dividiendo la suma por el contador:

```
contador=0
suma=0.0
numero=1.0        # sirve cualquiera mayor o igual a cero
while numero>=0:
    numero=float(input("número:"))
    contador=contador+1
    suma=suma+numero
promedio=suma/contador
print("el promedio es",promedio)
```

6) ¿Hay algún caso que nos obligue a salir del bucle, como por ejemplo para no procesar el último elemento? Sí. Tal como está el programa el último número introducido es negativo y no debería añadirse a la suma. Para resolverlo usaremos un condicional siempre con la misma expresión booleana del *while*, lo insertaremos en la zona del bucle donde sea necesario (inmediatamente después de la petición del dato en este caso) y desplazaremos a la derecha el resto del bucle para indentarlo (sangrarlo) y que sólo se ejecute si el número introducido es mayor o igual que cero:

```
contador=0
suma=0.0
numero=1.0        # sirve cualquiera mayor o igual a cero
while numero>=0:
    numero=float(input("número:"))
    if numero>=0: # siempre la misma expresión booleana del while
        contador=contador+1
        suma=suma+numero
promedio=suma/contador
print("el promedio es",promedio)
```

En este ejemplo puede además darse el caso de que no haya datos suficientes para calcular el promedio, es decir la parte del bucle donde se incrementa el contador y se añade un número a la suma no se ejecuta nunca si ya el primer dato es negativo. Esto lo debemos tener en cuenta cuando finaliza el bucle:

```
contador=0
suma=0.0
numero=1.0        # sirve cualquiera mayor o igual a cero
while numero>=0:
    numero=float(input("número:"))
    if numero>=0: # siempre la misma expresión booleana del while
        contador=contador+1
        suma=suma+numero
if contador==0:
    print("datos insuficientes para calcular el promedio")
else:
    promedio=suma/contador
    print("el promedio es",promedio)
```

Ésta es la forma preferida. Como curiosidad hay otras dos opciones. La primera sería emplear la instrucción *break* que existe en casi todos los lenguajes de programación y que nos permite salir del bucle. No hay que hacer el sangrado/indentación indicado antes y el condicional lleva la condición de terminación. El programa quedaría:

```
contador=0
suma=0.0
numero=1.0        # sirve cualquiera mayor o igual a cero
while numero>=0:
    numero=float(input("número:"))
    if numero<0:  # con break siempre la negada de la del while
        break
    contador=contador+1
    suma=suma+numero
if contador==0:
    print("datos insuficientes para calcular el promedio")
else:
    promedio=suma/contador
    print("el promedio es",promedio)
```

La otra opción consistiría en deshacer la última operación realizada, es decir si se sumó el último número que era negativo y se añadió una unidad al contador, habría que restar ese número de la suma y quitar una unidad al contador. Esta opción no siempre es posible, no es muy legible y por tanto no recomendable. El programa quedaría de esta forma:

```
contador=0
suma=0.0
numero=1.0        # sirve cualquiera mayor o igual a cero
while numero>=0:
    numero=float(input("número:"))
    contador=contador+1
    suma=suma+numero

# se deshacen las operaciones realizadas con el numero negativo
contador=contador-1
suma=suma-numero
if contador==0:
    print("datos insuficientes para calcular el promedio")
else:
    promedio=suma/contador
    print("el promedio es",promedio)
```

Ejemplo 3: Hacer un programa que calcule el producto de todos los pares entre dos enteros diferentes dados en orden creciente y ambos incluidos.

1) Sean n1 y n2 los enteros dados, el bucle debe repetirse un número determinado de veces dependiendo de lo que valgan n1 y n2. Será por tanto un bucle *for*.

```
???
for i in range(???):
    ???
    ???
???
```

2) Dentro del bucle hay que mirar si el número es par, y si lo es entonces multiplicarlo por una variable donde guardaremos el producto de todos los pares.

```
???
for i in range(???):
    if i%2==0:
        producto=producto*i
???
```

3) Si el índice del bucle debe tomar los valores entre n1 y n2 ambos incluidos el rango será

```
???
for i in range(n1,n2+1):
    if i%2==0:
        producto=producto*i
???
```

4) Hay que poner el valor inicial del producto, que debe ser 1. Si pusiéramos 0 siempre se multiplicaría todo por cero. También hay que poner valores para n1 y n2. En las etapas iniciales de creación del programa podemos poner asignaciones directas por rapidez. Esto es especialmente útil si son muchos los datos de entrada. En la versión final del programa no deben aparecer estas asignaciones directas:

```
#n1=int(input("número inicial:"))
#n2=int(input("número final:"))
n1=1
n2=4
producto=1.0
for i in range(n1,n2+1):
    if i%2==0:
        producto=producto*i
???
```

5) El resultado lo tenemos directamente:

```
#n1=int(input("número inicial:"))
#n2=int(input("número final:"))
n1=1
n2=4
producto=1.0
for i in range(n1,n2+1):
    if i%2==0:
        producto=producto*i
print("el producto es:",producto)
```

6) Podemos hacer pruebas con valores que produzcan resultados conocidos, en este caso el producto de 2 y 4. No hay que evitar ningún proceso con el último dato, el programa definitivo queda:

```
n1=int(input("número inicial:"))
n2=int(input("número final:"))
#n1=1
#n2=4
producto=1.0
for i in range(n1,n2+1):
    if i%2==0:
        producto=producto*i
print("el producto es:",producto)
```

Siempre podemos además en estas etapas iniciales de prueba añadir un *print* dentro del bucle para comprobar que la variable índice del bucle toma los valores deseados.

Este programa se podría haber hecho también recorriendo el bucle de dos en dos. Basta garantizar que el primer valor es par. Esta versión quedaría:

```
n1=int(input("número inicial:"))
n2=int(input("número final:"))
inicial=n1
if inicial%2==1:         # aseguramos que sea el primer par
    inicial=inicial+1
producto=1.0
for i in range(inicial,n2+1,2):
    producto=producto*i
print("el producto es:",producto)
```

En el enunciado se especifica que los números inicial y final están dados en orden creciente pero si no fuera así el bucle podría no ejecutarse y habría que añadir al final los condicionales correspondientes para indicar que el producto de cero valores es cero.

Ejemplo 4: Escribir un programa que calcule el máximo valor de n tal que la suma de todos los inversos hasta n es menor que un valor dado como dato.

Es decir, se pide calcular el valor n más alto tal que se cumple:

$$\sum_{i=1}^{i=n} \frac{1}{i} < v$$

1) La solución consiste en acumular la suma de inversos. No se sabe cuántos hay que sumar de manera que habrá que emplear un bucle *while*:

```
???
while ???:
    ???
    ???
???
```

2) Lo que hay que repetir es generar un nuevo *i* consecutivo del anterior y añadir su inverso a una suma:

```
???
while ???:
    i=i+1
    suma=suma+1.0/i
???
```

3) El bucle finaliza cuando esa suma sea mayor o igual que *v*, por tanto debe ejecutarse mientras la suma sea menor que *v*:

```
???
while suma<v:
    i=i+1
    suma=suma+1.0/i
???
```

4) La suma hay que ponerla a cero. Si el primer sumando es el inverso de 1 entonces hay que inicializar *i* con el valor cero.

```
suma=0.0
i=0
while suma<v:
    i=i+1
    suma=suma+1.0/i
???
```

5) Se sale del bucle cuando ya hemos alcanzado o excedido el valor dado, por tanto el valor de *n* pedido no es el último sino el anterior al último:

```
suma=0.0
i=0
while suma<v:
    i=i+1
    suma=suma+1.0/i
n=i-1
```

6) No hay que ejecutar nada condicionado luego basta pedir *v* por teclado y mostrar *n* por la salida:

```
v=float(input("valor de v:"))
suma=0.0
i=0
while suma<v:
    i=i+1
    suma=suma+1.0/i
n=i-1
print("el n pedido es:",n)
```

Aunque es ésta la versión definitiva del programa, siempre conviene hacer pruebas con valores cuyo resultado se conozca. Por ejemplo si la suma de 1/1 y 1/2 es 1.5 y el siguiente término es $0.\hat{3}$ entonces podemos elegir 1.6 como un buen valor de prueba. Nos tiene que salir un 2 como resultado. Es posible hacer una traza de la evolución del bucle partiendo de los valores iniciales:

i	suma	suma<1.6	n
	0.0		
0			
		True	
1			
	1.0		
		True	
2			
	1.5		
		True	
3			
	$1.8\hat{3}$		
		False	
			2

¿Podía haberse hecho de otra forma con este mismo procedimiento? Sí, por ejemplo el paso 2 alguien puede decidir escribirlo así:

```
???
while ???:
    suma=suma+1.0/i
    i=i+1
???
```

El paso 3) sería exactamente igual puesto que la condición de terminación es la misma pero en el paso 4) los valores iniciales serían ahora distintos. *i* debe tomar inicialmente el valor de 1. Además, como el incremento se realiza después de que la suma exceda el límite, el valor de *n* pedido es ahora *i-2*. El programa final quedaría es este caso:

```
v=float(input("valor de v:"))
suma=0.0
i=1
while suma<v:
    suma=suma+1.0/i
    i=i+1
n=i-2
print("el n pedido es:",n)
```