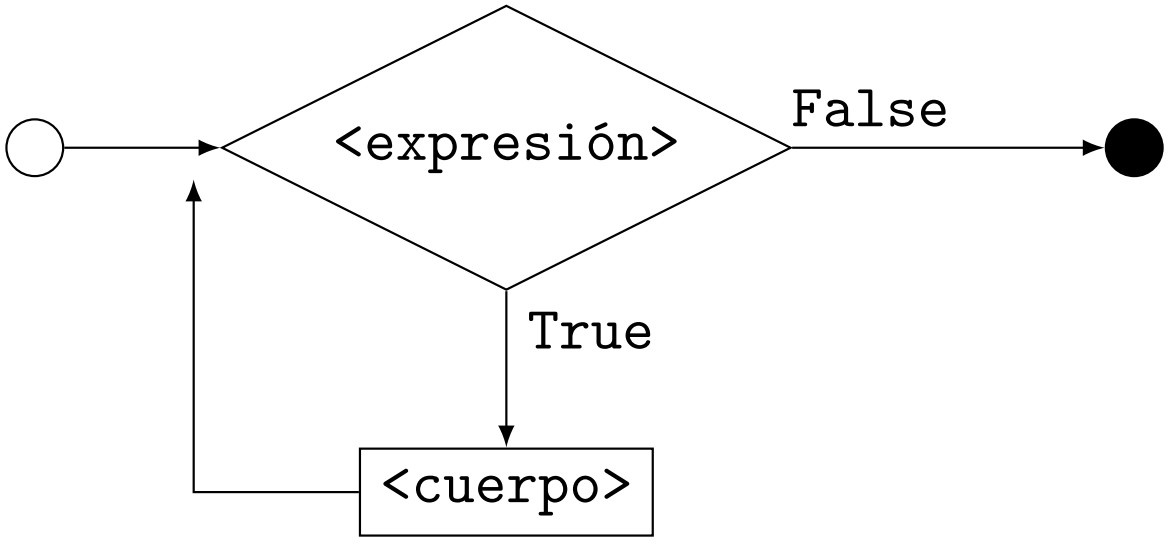
Ciclos interactivos y ciclos con centinela

Introducción a la Programación

Departamento de Ciencias Básicas, UNLu

Ciclos indefinidos WHILE

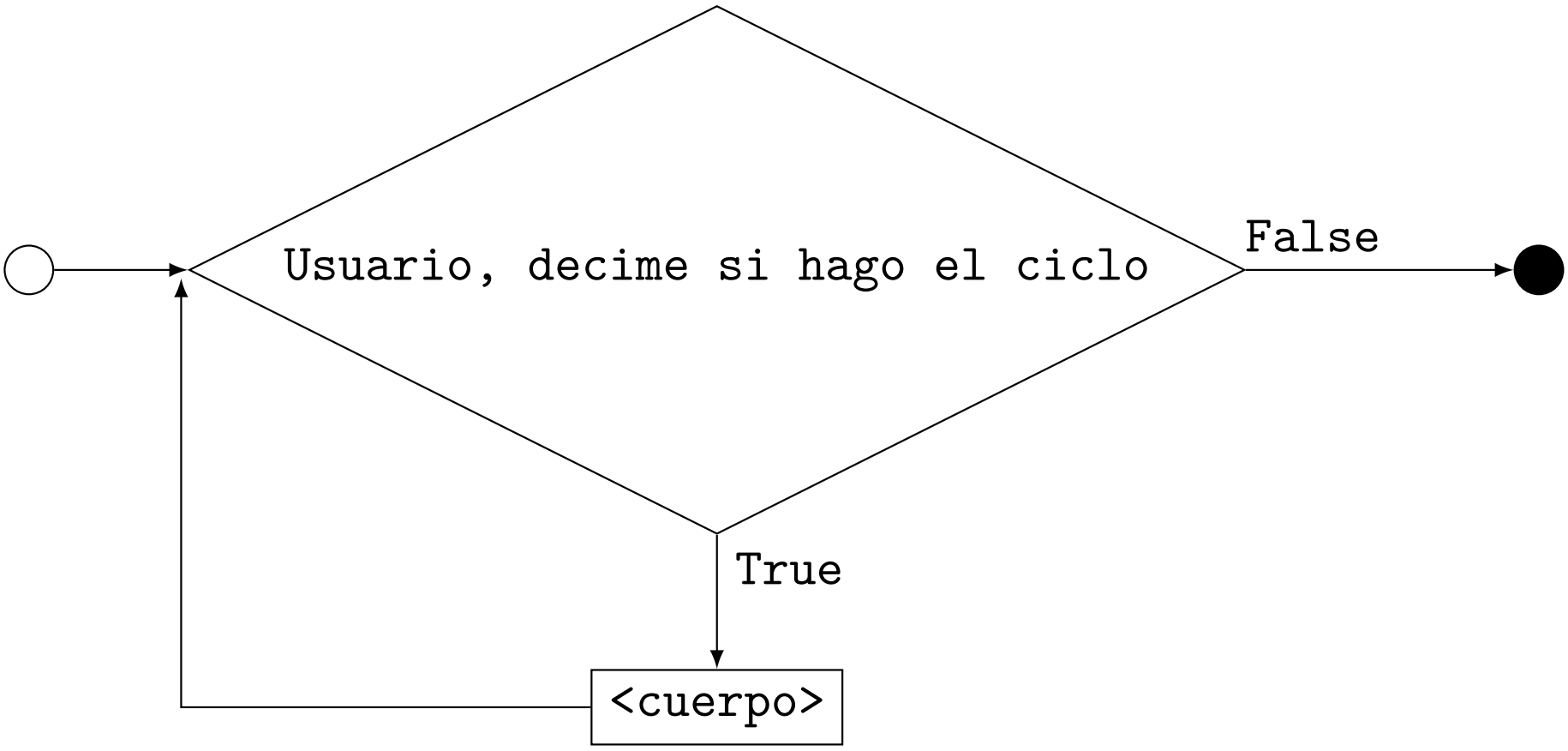
1. Evaluar la condición.
2. Si la condición es falsa, salir del ciclo.
3. Si la condición es verdadera, ejecutar el cuerpo, y volver al paso 1.



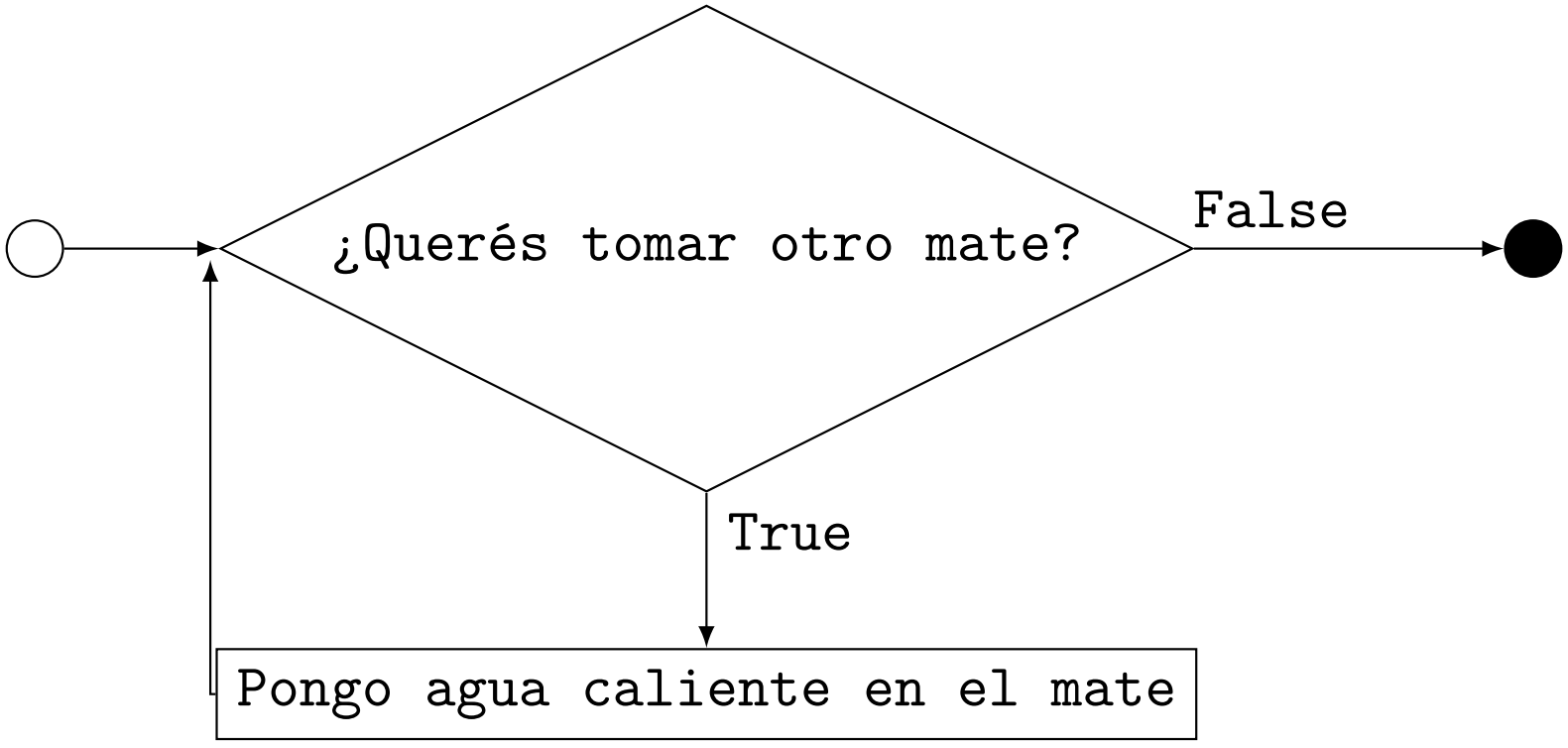
Ciclo interactivo

Es simplemente un ciclo indefinido, con la particularidad de que interactuamos con el usuario antes de hacer cada iteración del cuerpo del ciclo, para saber si debemos ejecutar el cuerpo del ciclo, o se terminó el ciclo.

Ciclo interactivo



Ejemplo ciclo interactivo "la mateada"



Ciclo con centinela

Es simplemente un ciclo indefinido, con la particularidad de que antes de hacer cada iteración del cuerpo del ciclo nos fijamos si la variable centinela tiene cierto valor distinguido, para saber si debemos ejecutar el cuerpo del ciclo, o se terminó el ciclo.

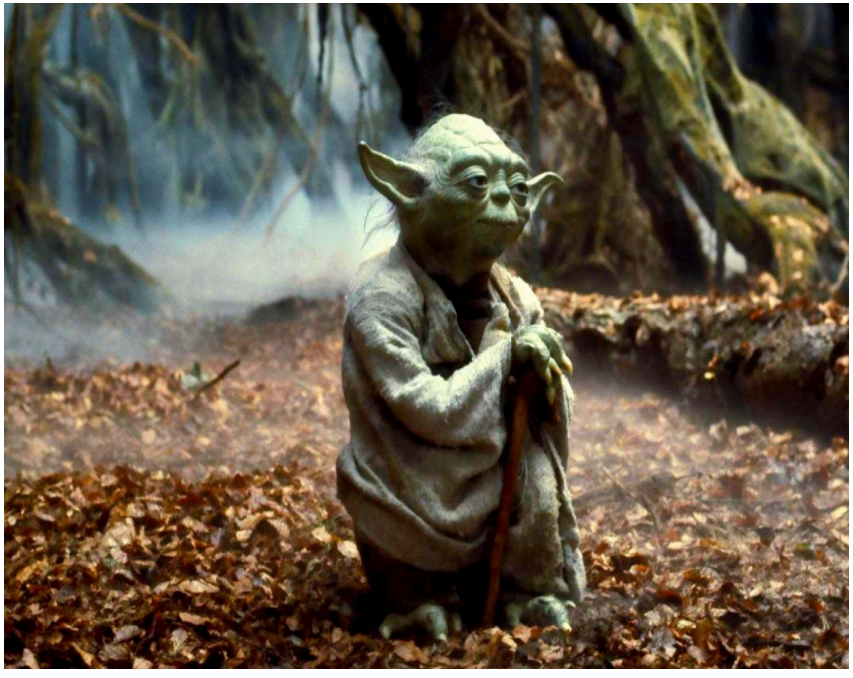
Ejemplo de ciclo con centinela en el super

Ejemplo de ciclo con centinela en el super

La persona que atiende la caja en el supermercado, agarra cada uno de los productos y carga el precio en la máquina registradora, hasta que llega al separador plástico, que le indica que los productos que siguen son del próximo cliente.

Moraleja

Le podemos decir al cajer@ del super de venir a estudiar sistemas a la UNLu ya que ya sabe uno de los temas más difíciles de Introducción a la Programación!



Ejercicio (del apunte)

El usuario debe poder ingresar muchos números y cada vez que se ingresa uno debemos informar si es positivo, cero o negativo.

Entrada-Salida

Para resolverlo, reflexionamos sobre el problema. Lo primero que debemos resolver es: Cuál es la entrada? Cuál es la salida?

Entrada: 1) Algún número ingresado. 2) Alguna indicación de si se esperan más ingresos de números o terminamos.

Salida: Informar si el número es positivo, cero, o negativo. Como debemos informarlo podríamos simplemente imprimir por pantalla un string (en este caso, no hay salida). Otra posibilidad sería hacer una función que retorne un string, e imprimir el string en el cuerpo principal del programa.

Pseudocódigo

Escribimos un pseudocódigo de la solución (la flechita ← en pseudocódigos denota asignación):

```
hay_mas_datos ← "Sí"
Mientras hay_mas_datos sea igual a "Sí":
    Pedir datos del número
    Realizar cálculos de cero, positivo, negativo
    Preguntar al usuario si desea ingresar más datos
    ("Sí" cuando quiere ingresar más datos)
```

Solución con Ciclo Interactivo

Pasamos nuestro pseudocódigo a código Python:

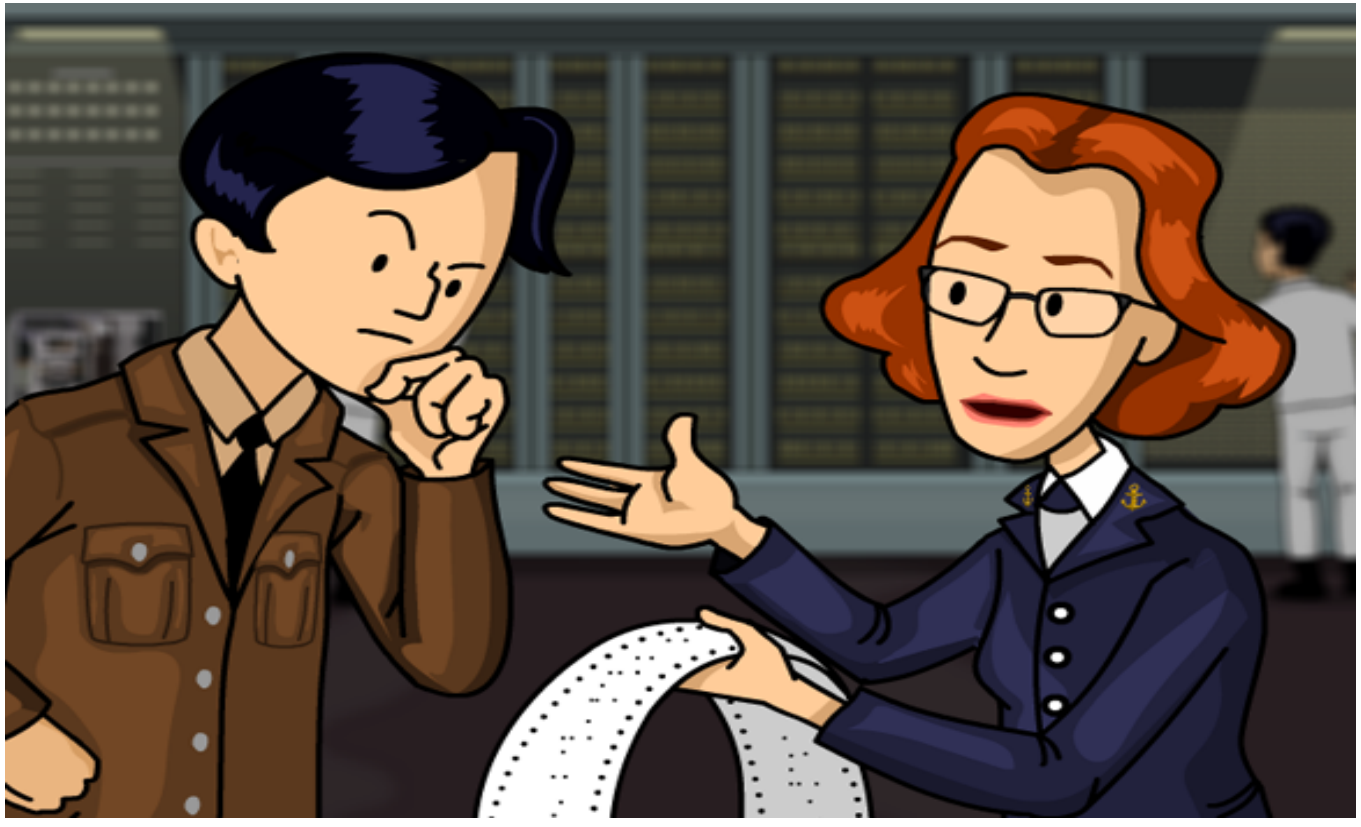
```
def muchos_pcn():
    hay_mas_datos = "Si"
    while hay_mas_datos == "Si":
        x = int(input("Ingrese un numero: "))
        if x > 0:
            print("Numero positivo")
        elif x == 0:
            print("Igual a 0")
        else:
            print("Numero negativo")

        hay_mas_datos = input("Quiere seguir? <Si-No>: ")
```

Ya terminamos?? NOOOOOO!!!

¡HAY QUE TESTEARLO!

Caso de test

Un caso de test es un par: (entrada, salida esperada).



Testeando: particionando los inputs

Particionamos las posibles entradas en subconjuntos con distintas características, que representen las diversas clases de entrada. En cada una de estas particiones, los elementos tendrán características similares. De cada una de estas particiones, tomaremos elementos representantivos, con los cuales compondremos los casos de test.

- Números positivos
- Cero
- Números negativos

Importante: Con el testing, no podemos garantizar que no haya errores (para ello deberíamos testear sobre todos los posibles inputs), pero tenemos menos errores.

Testeando: haciendo los casos de tests

Como en nuestro código, imprimimos por pantalla el resultado, no hay salida! Algunos casos posibles de tests podrían ser:

- Números positivos: (1, mirar la pantalla si imprime "Numero positivo")
- Cero: (0, mirar la pantalla si imprime "Igual a 0")
- Números negativos: (-1, mirar la pantalla si imprime "Numero negativo").

Como vemos, este caso es muy simple, pero si hiciéramos muchos casos de tests, para testearlo deberíamos mirar o contratar una persona que mire largamente la pantalla.

Testeando: haciendo los casos de tests

Nos hubiera convenido hacer una función, que retorne algo, así podemos hacer casos de test automatizados. En lugar de sólo imprimir por pantalla el resultado, mejor hacer una función que retorne el string, y luego que el cuerpo principal del programa imprima el resultado.
De esa forma, algunos casos posibles de tests podrían ser:

- Números positivos: (1, "Numero positivo")
- Cero: (0, "Igual a 0")
- Números negativos: (-1, "Numero negativo").

No haber pensado los casos de tests antes de codificar, nos obliga a volver a codificar, y esto puede ser muy costoso en tiempo y dinero.

Reescribiendo el código para que se pueda testear automáticamente

```python
def pcn(x):
    if x >= 0:
        resultado = "Numero positivo"
    elif x == 0:
        resultado = "Igual a 0"
    else:
        resultado = "Numero negativo"
    return resultado

def muchos_pcn():
    hay_mas_datos = "Si"
    while hay_mas_datos == "Si":
        x = int(input("Ingrese un numero: "))
        print(pcn(x))
        hay_mas_datos = input("Quiere seguir? <Si-No>: ")

muchos_pcn()
```

Tests simples en Ptyhon

```python
def pcn(x):
    if x >= 0:
        resultado = "Numero positivo"
    elif x == 0:
        resultado = "Igual a 0"
    else:
        resultado = "Numero negativo"
    return resultado

def muchos_pcn():
    hay_mas_datos = "Si"
    while hay_mas_datos == "Si":
        x = int(input("Ingrese un numero: "))
        print(pcn(x))
        hay_mas_datos = input("Quiere seguir? <Si-No>: ")

#muchos_pcn()

def test_pcn():
    assert pcn(0) == "Igual a 0", "Debe ser igual a 0"
    assert pcn(1) == "Numero positivo", "Debe ser un numero positivo"
    assert pcn(10) == "Numero positivo", "Debe ser un numero positivo"
    assert pcn(-1) == "Numero negativo", "Debe ser un numero negativo"
    assert pcn(-10) == "Numero negativo", "Debe ser un numero negativo"
    print("Función pcn testeada exitosamente.")

test_pcn()
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
  File "c:\Users\rmatu\Documents\GitHub\pcnbug.py", line 20, in test_pcn
    assert pcn(0) == "Igual a 0", "Debe ser igual a 0"
AssertionError: Debe ser igual a 0
```

Bug corregido: pasamos los tests!

```python
def pcn(x):
    if x > 0:
        resultado = "Numero positivo"
    elif x == 0:
        resultado = "Igual a 0"
    else:
        resultado = "Numero negativo"
    return resultado

def muchos_pcn():
    hay_mas_datos = "Si"
    while hay_mas_datos == "Si":
        x = int(input("Ingrese un numero: "))
        print(pcn(x))
        hay_mas_datos = input("Quiere seguir? <Si-No>: ")

#muchos_pcn()

def test_pcn():
    assert pcn(0) == "Igual a 0", "Debe ser igual a 0"
    assert pcn(1) == "Numero positivo", "Debe ser un numero positivo"
    assert pcn(10) == "Numero positivo", "Debe ser un numero positivo"
    assert pcn(-1) == "Numero negativo", "Debe ser un numero negativo"
    assert pcn(-10) == "Numero negativo", "Debe ser un numero negativo"
    print("Función pcn testeada exitosamente.")

test_pcn()
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
  File "c:\Users\rmatu\Documents\GitHub\pcnbug.py", line 20, in test_pcn
    assert pcn(0) == "Igual a 0", "Debe ser igual a 0"
AssertionError: Debe ser igual a 0
PS C:\Users\rmatu>  ${env:DEBUGPY_LAUNCHER_PORT}='57403'; & 'C:\Users\rmatu\AppData\Local\Programs\Python\Python38-
s\ms-python.python-2020.4.74986\pythonFiles\lib\python\debugpy\no_wheels\debugpy\launcher' 'c:\Users\rmatu\Document
Función pcn testeada exitosamente.
```

Problema con el ciclo interactivo

Un problema que tiene nuestra solución del ciclo interactivo es que resulta poco amigable preguntarle al usuario después de cada cálculo si desea continuar.

Para evitar esto, se puede usar el método del centinela: un valor arbitrario que, si se lee, le indica al programa que el usuario desea salir del ciclo.

Podemos suponer que si el usuario ingresa el carácter *, es una indicación de que desea terminar (pero podríamos haber elegido otro valor distinguido para el centinela, como el carácter % o el string "Listo").

Pseudocódigo

Escribimos un pseudocódigo de la solución...

```
centinela ← Pedir datos
Mientras centinela no sea igual a "*":
  Realizar cálculos de cero, positivo, negativo
  centinela ← Pedir datos
```

... y después simplemente pasamos nuestro pseudocódigo a código Python.

Solución con Ciclo con Centinela

```python
def pcn(x):
    if x > 0:
        resultado = "Numero positivo"
    elif x == 0:
        resultado = "Igual a 0"
    else:
        resultado = "Numero negativo"
    return resultado

def muchos_pcn():
    centinela = input("Ingrese un numero (* para terminar): ")
    while centinela != "*":
        x = int(centinela)
        print(pcn(x))
        centinela = input("Ingrese un numero (* para terminar): ")

muchos_pcn()
```

Mejorando nuestro código

El código de nuestro ciclo con centinela tiene un problema: las líneas 11 y 15 están repetidas!

Si en la etapa de mantenimiento tuviéramos que realizar un cambio en el ingreso del dato (por ejemplo, cambiar el mensaje) deberíamos estar atentos y corregir ambas líneas.

¿Cómo eliminar el código duplicado? Una opción es extraer el código duplicado en una función.

Hacer estas mejoras que no alteran la funcionalidad del código, pero que lo mejoran, se denomina "refactoring".

Mejorando el código (refactoring)

```
1   def pcn(x):
2       if x > 0:
3           resultado = "Numero positivo"
4       elif x == 0:
5           resultado = "Igual a 0"
6       else:
7           resultado = "Numero negativo"
8       return resultado
9
10  def leer_centinela():
11      return input("Ingrese un numero (* para terminar): ")
12
13  def muchos_pcn():
14      centinela = leer_centinela()
15      while centinela != "*":
16          x = int(centinela)
17          print(pcn(x))
18          centinela = leer_centinela()
19
20  muchos_pcn()
```

Cómo resolver un problema computacional

1. Entender cuál es el problema que me piden resolver
2. Cuáles son los posibles inputs?
3. Cuál es el output?
4. Cuál es la relación entre la entrada y la salida? Hacer ejemplos sencillos para entender dicha relación. Estos ejemplos nos servirán luego también para testear que lo que hicimos es correcto.
5. ¿Cómo resolveríamos el problema si no tuviésemos ninguna computadora?
6. Hacer algoritmo (pseudocódigo)
7. Programar
8. Testear
9. Refactoring