Ciclos y Rangos

Unidad 3

Apunte de cátedra

Pensamiento computacional (90)
Cátedra: Camejo

.UBAXXI

Ciclos y Rangos

Supongamos que al fin consigues tu primer empleo en una fábrica cerca de tu casa. Uno de los primeros trabajos que te asigna tu jefe es escribir unos procedimientos para facilitar el entrenamiento de nuevo personal. Así que, para comenzar te encarga la descripción de uno muy simple: descarga de materias primas frágiles XX21 del camión del proveedor y almacenamiento en el depósito. Las condiciones de la tarea son bien sencillas. Siempre se encarga la tarea a un único operario y en el camión suelen venir entre 5 y 15 cajas del material. Así que pones manos a la obra y produces tu propia página para el manual de procedimientos de la empresa. Comienzas describiendo la operación para la descarga de 5 cajas. Más o menos te quedaría así:

1 Abrir la puerta del depósito y encender luces
2 Ir al garage o playón donde estacionó el camión
3 Abrir las puertas traseras de la caja del transporte
4 Tomar una caja con ambas manos, asegurándola para no tirarla
5 Caminar sosteniendo la caja hasta el depósito
6 Colocar la caja sobre el piso en el sector correspondiente
7 Ir al garage o playón donde estacionó el camión
8 Tomar una caja con ambas manos, asegurándola para no tirarla
9 Caminar sosteniendo la caja hasta el depósito
10 Colocar la caja sobre la caja anterior
11 Ir al garage o playón donde estacionó el camión
12 Tomar una caja con ambas manos, asegurándola para no tirarla
13 Caminar sosteniendo la caja hasta el depósito
14 Colocar la caja sobre la caja anterior
15 Ir al garage o playón donde estacionó el camión
16 Tomar una caja con ambas manos, asegurándola para no tirarla
17 Caminar sosteniendo la caja hasta el depósito
18 Colocar la caja en el piso a la derecha de la pila recién armada
19 Ir al garage o playón donde estacionó el camión
20 Tomar una caja con ambas manos, asegurándola para no tirarla
21 Caminar sosteniendo la caja hasta el depósito
22 Colocar la caja sobre la caja anterior
23 Apagar luces y cerrar puerta del depósito
24 Ir al garage o playón donde estacionó el camión
25 Cerrar y trabar puertas del camión
26 Avisar fin de descarga al transportista

Nada mal, no? Se lo muestras a tu jefe. Te da el ok y te pide que definas el mismo procedimiento para todos los casos posibles (recuerda que pueden venir entre 5 y 15 cajas en el camión). ¿Qué vas a hacer? ¿Vas a incorporar una página casi igual por cada posible cantidad de paquetes por remesa?¿Y si algún día vienen

más de 15 o menos de 5? ¿Y si en la descarga anterior se rompió material y en esta envían una caja de más para restituir la dañada? Cada nuevo empleado que entre a la fábrica tendrá que leerse 11 páginas (que contengan casi lo mismo; no hay posibilidad de que se concentre) y deberá pedir instrucciones, de todas formas, a su supervisor siempre que venga un número no previsto de cajas. Suena a nada de Ingeniería. ¿Por qué no ensayar algo más genérico que le facilite la vida a todos? Una nueva versión:

1 Abrir la puerta del depósito y encender luces
2 Ir al garage o playón donde estacionó el camión
3 Abrir las puertas traseras de la caja del transporte
4 Tomar una caja con ambas manos, asegurándola para no tirarla
5 Caminar sosteniendo la caja hasta el depósito
6 Si es la primera caja, colocarla sobre el piso en el sector correspondiente; si no, apilarla sobre la anterior; salvo que ya haya 3 apiladas, en ese caso colocarla a la derecha sobre el piso
7 Ir al garage o playón donde estacionó el camión
8 Repetir 4,5,6,7 mientras queden cajas para descargar
9 Cerrar y trabar puertas del camión
10 Avisar fin de descarga al transportista
11 Volver a depósito
12 Apagar luces y cerrar puerta del depósito

Mejor, no? Pensemos, esta descripción es bastante más compacta y cubre todas las posibles cantidades de cajas en un envío (habituales y excepcionales), de modo que con una única página en el manual de procedimientos será suficiente (el planeta, los nuevos empleados y la oficina de procedimientos agradecidos) y parece bastante clara y simple para cualquier novato. Ciertamente esta es una versión superadora!

Sin embargo, los algoritmos de los programas que vienes escribiendo hasta ahora se parecen más al primer intento de procedimiento. Hora de tomar de nuestra propia medicina y aplicarle Ingeniería a nuestros programas.

Para este cometido necesitamos contar con una herramienta especial que nos permita escribir, de manera compacta, la ejecución de ciertos pasos de forma repetida. Esa clase de recurso se denomina genéricamente **Ciclo** , **Bucle** o **Sentencia Iterativa**; y es una máquina repetidora de pasos o instrucciones en un programa. Es decir que podemos elegir una parte de nuestro programa, que necesitamos se ejecute repetidamente y de forma contigua una cantidad fija o variable de veces, y colocarla dentro de una sentencia especial con ese objetivo; que funciona como un pequeño motorcito encargado de realizar esa repetición automáticamente, una y otra vez, hasta terminar. Las ubicamos dentro del grupo de las **Estructuras de Control** (como los Condicionales) ya que es una sentencia estructurada, con encabezado y cuerpo; y tienen la capacidad de alterar el **Flujo de Control** Normal de un **Programa** (**FCP**).

Los **Bucles** son primitivas sencillas, normalmente tienen una sintaxis simple. Su dificultad no está en cómo se escriben, ni en entender su funcionamiento, sino en la manera en que se ajustan sus parámetros. Esto es lo

que nos garantizará que haga exactamente lo que esperamos. Tenemos que prestar atención tanto al código que debe repetir como a la cantidad de veces exacta que se repetirá (no sólo si conocemos esa cantidad cuando se escribe el **Ciclo**, es decir cantidad **constante** de repeticiones, sino también en el caso de posponer el conocimiento de esa cifra al momento de ejecución, cantidad **variable** de repeticiones).

Todos los Lenguajes de tipo Procedural disponen de una o más sentencias de este tipo. Los componentes de un **Ciclo**, en términos generales son:

- **Invariante**
- **Cuerpo**

La **Invariante** es la parte de la sentencia que permite decidir si se ejecutará el **Cuerpo** de ese **Bucle** una vez más o se abandonará el mismo para proseguir con la sentencia siguiente al **Ciclo**. Normalmente se escribe como una **Condición** válida. Hay que tener en cuenta que posee doble función, determina cuándo abrir la puerta de ingreso para que el **Cuerpo** sea ejecutado y cuándo cerrarla definitivamente.

El **Cuerpo** es la parte del **Bucle** que se ejecutará repetidamente. Un conjunto de sentencias válidas que se ejecutan una y otra vez (**como conjunto, siguiendo el flujo interno**) hasta que la evaluación de la **Condición** (que es la llave del **Bucle**) o **Invariante** diga lo contrario.

¿Cómo funciona un **Ciclo**?

En la gráfica observamos cómo funciona, en términos generales, un **Bucle**. La denominación de **Bucle** deriva de la forma similar a un bucle o rulo que produce el **FCP** al repetir el **Cuerpo** (claro que a mano alzada se parecía más, pero el nombre genérico es anterior a las Herramientas de Graficación Digitales que usamos hoy para diseñar Diagramas de Flujo)

Hay varias versiones de **Bucle**, pero la del esquema es la más genérica y extendida, y coincide con la interpretación de **Ciclo** de las sentencias de Python que usaremos.

¿Cómo funciona entonces?

Cuando el **FCP** llega a un **Ciclo**, lo primero que hace es evaluar la **Condición** (recordá: llave de entrada/salida). Si la **Condición** da Verdadero se ejecuta completamente el **Cuerpo**; y cuando se termina de ejecutar el mismo, el **FCP** vuelve a la **Condición** (dibujando el lazo, rulo o bucle) y esta se evalúa nuevamente. Si vuelve a dar Verdadero se repite todo el proceso anterior; si en cambio da Falso el **FCP** abandona el **Ciclo** y ejecuta la sentencia siguiente al **Bucle**.

Es decir, el programa llega, entra (si se dan las condiciones) y se queda iterando allí (repitiendo el **Cuerpo**) todas las veces que haga falta. Como en una calesita, cuando las condiciones determinan el fin de las repeticiones se va del **Ciclo**, y no vuelve.

Al escribir un **Bucle** necesitamos tener muy en claro 4 aspectos relevantes. Si nos equivocamos en cualquiera de ellos el **Ciclo** no funcionará como esperamos. Estos 4 aspectos son los siguientes:

- **Cuerpo**: Bloque de sentencias que se ejecuta por vuelta
- **Condición o Invariante**: Puerta de entrada/salida al **Ciclo**
- **Estado Previo**: Valores de las variables de la **Condición** antes del **Bucle**
- **Paso**: Posibilidad de cambio de valor en alguna de las variables de la **Condición**

Vamos a resolver un problema simple usando el más básico de los **Bucles** de Python: **while**; y veremos qué significan cada uno de estos aspectos.

Sintaxis de **while**

```
while condición:
        -
        -
        -
```

¿Cómo se ejecuta **while** de Python?

Cuando el **FCP** llega a una sentencia **while** lo primero que se realiza es la evaluación de la **Condición**. Si la **Condición** da VERDADERO se ejecutará completamente el **Bloque de Sentencias** del **Cuerpo** del **while**. Cuando termina la ejecución del **Cuerpo** el **FCP** vuelve a la **Condición** y la evalúa nuevamente. Si da VERDADERO repite lo anterior; si da FALSO sale del **while** y ejecuta la sentencia siguiente al mismo. Si la primera vez que el **FCP** llega al **while** y evalúa la **Condición** da FALSO, el **FCP** sigue de largo a la próxima sentencia y saltea todo el **Cuerpo**.

Haremos un programa que pida 5 números y cuente cuántos múltiplos de 5 vinieron.

```
# Contaremos cuántos múltiplos de 5 ingresan en un lote de 5 números
print('Ingresá 5 Números enteros')
totalMult=0
```

```
num=int(input('Número: '))
if num%5==0:     #identifico si es múltiplo de 5
    totalMult+=1
num=int(input('Número: '))
if num%5==0:
    totalMult+=1
num=int(input('Número: '))
if num%5==0:
    totalMult+=1
num=int(input('Número: '))
if num%5==0:
    totalMult+=1
num=int(input('Número: '))
if num%5==0:
    totalMult+=1
print('Vinieron:',totalMult,'múltiplos de 5')
```

Es fácil ver que este programa es un candidato serio para empleo de **Bucle**. Hay un conjunto de instrucciones que se repiten de manera contigua. También podemos razonar que si quisiéramos extender la cantidad de números sobre los que va a trabajar, deberíamos hacer algunos copy-paste del grupo de sentencias. Y si decidimos preguntarle al usuario cuántos números ingresará estamos fritos. Ya que no podremos anticipar cuántos bloques repetiremos, no queda otra que emplear **Ciclos**. De todos modos, la manera en que está planteado nos ayuda para la implementación de una sentencia iterativa. Podríamos haber solicitado todos los números al principio y haber construido un árbol de condiciones para averiguar cuántos de dichos datos eran múltiplos de 5. Se hace evidente que cuando disponemos de una herramienta es importante seleccionar una posible solución que sea fácilmente adaptable para el empleo de la misma y no que entorpezca su uso. Esto también forma parte de la dificultad en el uso de **Ciclos**, debes aprender a pensar algoritmos aptos para su uso.

Dijimos que para armar y afinar correctamente una Iteración teníamos que prestar atención a 4 aspectos:

- **Cuerpo**
- **Condición**
- **Estado Previo**
- **Paso**

Primero vamos a pensar cuál sería el **Cuerpo**, o al menos, el mínimo **Cuerpo**. El **Cuerpo** será, sin dudas, la porción del código que se repetirá de forma contigua. En nuestro ejemplo **α**.

```
# Contaremos cuántos múltiplos de 5 ingresan en un lote de 5 números
print('Ingresá 5 Números enteros')
totalMult=0
num=int(input('Número: '))
if num%5==0:     #identifico si es múltiplo de 5
    totalMult+=1
num=int(input('Número: '))
if num%5==0:
```

```
    totalMult+=1
num=int(input('Número: '))
if num%5==0:
    totalMult+=1
num=int(input('Número: '))
if num%5==0:
    totalMult+=1
num=int(input('Número: '))
if num%5==0:
    totalMult+=1
print('Vinieron:',totalMult,'múltiplos de 5')
```

α

Así que empezaremos por reescribir el programa compactándolo:

```
# Contaremos cuántos múltiplos de 5 ingresan en un lote de 5 números
print('Ingresá 5 Números enteros')
totalMult=0

    num=int(input('Número: '))
    if num%5==0:     #identifico si es múltiplo de 5
        totalMult+=1
print('Vinieron:',totalMult,'múltiplos de 5')
```

Nuestro **Cuerpo** se posiciona desplazado a la derecha, ya que debe ir indentado. Pero nos falta el encabezado. Para eso debemos pensar en la **Condición** que sirva de puerta vaivén. Debe poder abrirse para ingresar a ejecutar el **Cuerpo** todas las veces que haga falta y cerrarse definitivamente cuando ya repetimos el mismo las veces necesarias. Así que pensemos, debemos pedir un número y averiguar si es múltiplo de 5 unas 5 veces. O, lo que es lo mismo, hasta que no hayamos hecho 5 veces esto (5 vueltas) no paramos. Miremos esta posible condición, entonces:

```
# Contaremos cuántos múltiplos de 5 ingresan en un lote de 5 números
print('Ingresá 5 Números enteros')
totalMult=0
while veces<=5:
    num=int(input('Número: '))
    if num%5==0:     #identifico si es múltiplo de 5
        totalMult+=1
print('Vinieron:',totalMult,'múltiplos de 5')
```

Perfecto! Ejecutemos esta versión y veamos cómo nos va.

Ok, si ya hiciste la prueba, te habrás dado cuenta que a Python no le gustó nuestro programa y nos arrojó afuera con un error fatal. ¿Qué sucedió? Algo muy sencillo, no aseguramos el **Estado Previo**. Es decir, ***no preparamos con algún valor controlado cada una de las variables que forman parte de la Condición antes de comenzar a ejecutar el Bucle***. En nuestro caso, la **Condición** tiene una única variable **veces** y como no estuvo previamente inicializada cuando le preguntamos a Python por su valor enloquece y pierde los

modales. Así que será necesario darle un valor inicial. ¿Dónde? En cualquier punto previo al encabezado del **while**. Veamos si podemos subsanar el problema:

```
# Contaremos cuántos múltiplos de 5 ingresan en un lote de 5 números
print('Ingresá 5 Números enteros')
totalMult=0
veces=0
while veces<=5:
    num=int(input('Número: '))
    if num%5==0:     #identifico si es múltiplo de 5
        totalMult+=1
print('Vinieron:',totalMult,'múltiplos de 5')
```

Hagamos la prueba. Ahora a Python le gusta nuestro programa, tanto que no para de pedirnos números… Tal parece que no acabaremos nunca y sólo nos queda acabar la ejecución abruptamente cerrando la ventana o con Ctrl-C ¿Qué sucedió? Entramos en lo que se conoce como un **Loop Infinito**, una calesita que no para nunca. ¿Por qué sucedió esto? Simple: no cuidamos el **Paso** o **Avance**. Y cómo lo logramos? Tenemos que asegurarnos que dentro del **Cuerpo**, ***por lo menos para una de las variables de la condición, haya al menos una sentencia donde se le pueda cambiar el valor***. En criollo: al menos una de las variables de la Condición debe aparecer al menos en alguna asignación a la **izquierda** dentro del **Cuerpo**. De ese modo, eventualmente el cambio en el valor de la variable puede producir un valor **False** de la **Condición**.

En nuestro problema, si vamos contando cada número ingresado, en algún momento la cuenta nos dará un valor mayor que 5. Observá cómo podría quedar:

```
# Contaremos cuántos múltiplos de 5 ingresan en un lote de 5 números
print('Ingresá 5 Números enteros')
totalMult=0
veces=0
while veces<=5:
    num=int(input('Número: '))
    if num%5==0:     #identifico si es múltiplo de 5
        totalMult+=1
    veces+=1
print('Vinieron:',totalMult,'múltiplos de 5')
```

Como **veces** no se usa para nada dentro del **Cuerpo**, en realidad su actualización podría darse en cualquier parte dentro del mismo. Antes del ingreso del número, o antes del Condicional también.

¿Andará nuestro programa por fin? Casi casi. Falta un pequeño detalle, si lo probamos observaremos que en realidad nos pide 6 números en lugar de 5. Este tipo de desajustes son comunes, incluso si sos un programador experimentado; pero se salvan fácilmente. Debes ajustar el **Estado Previo**, la **Condición**, el **Paso**, o cualquier combinación de ellos, hasta que el combo funcione. Por ejemplo, sólo tocando la incialización de **veces** podemos hacer que funcione correctamente:

```
# Contaremos cuántos múltiplos de 5 ingresan en un lote de 5 números
```

```
print('Ingresá 5 Números enteros')
totalMult=0
veces=1
while veces<=5:
    num=int(input('Número: '))
    if num%5==0:     #identifico si es múltiplo de 5
        totalMult+=1
print('Vinieron:',totalMult,'múltiplos de 5')
```

Pero te pongo otros ejemplos que también funcionarán (sólo por mencionar unos pocos)

```
# Contaremos cuántos múltiplos de 5 ingresan en un lote de 5 números
print('Ingresá 5 Números enteros')
totalMult=0
veces=0
while veces<5:
    num=int(input('Número: '))
    if num%5==0:     #identifico si es múltiplo de 5
        totalMult+=1
    veces+=1
print('Vinieron:',totalMult,'múltiplos de 5')
```

```
# Contaremos cuántos múltiplos de 5 ingresan en un lote de 5 números
print('Ingresá 5 Números enteros')
totalMult=0
veces=5
while veces>0:
    num=int(input('Número: '))
    if num%5==0:     #identifico si es múltiplo de 5
        totalMult+=1
    veces-=1
print('Vinieron:',totalMult,'múltiplos de 5')
```

```
# Contaremos cuántos múltiplos de 5 ingresan en un lote de 5 números
print('Ingresá 5 Números enteros')
totalMult=0
veces=10
while veces<=50:
    num=int(input('Número: '))
    if num%5==0:     #identifico si es múltiplo de 5
        totalMult+=1
    veces+=10
```

```
print('Vinieron:',totalMult,'múltiplos de 5')
```

¿Cómo quedaría armado nuestro Bucle con el resto de los ajustes importantes?

```
# Contaremos cuántos múltiplos de 5 ingresan en un lote de 5 números
print('Ingresá 5 Números enteros')
totalMult=0
veces=1
while veces<=5:
    num=int(input('Número: '))
    if num%5==0:      #identifico si es múltiplo de 5
        totalMult+=1
    veces+=1
print('Vinieron:',totalMult,'múltiplos de 5')
```


Sin un **Estado Previ**o controlado existen serias posibilidades de que el **Bucle** nunca se ejecute, aún cuando debiera, o que se produzca un error al preguntar por el valor de una variable desconocida. Sin una **Condición** correcta es probable que la cantidad de vueltas no sea la adecuada. Sin un **Paso** seguramente entraríamos en un loop infinito, es decir, nunca podemos salir del **Bucle**. Sin una adecuada combinación de todos ellos probablemente la cantidad de vueltas no sea apropiada.

¡Advertencias!

✔ *La primera vez que referencies una variable en un programa SIEMPRE debe estar a la IZQUIERDA de una asignación! De ese modo evitarás intentar consultar un dato desconocido. Error semántico de los más comunes.*

✔ *Es bastante frecuente, aún con práctica, que el **Bucle** pueda quedarte desajustado con respecto a la cantidad de vueltas. Sólo debes probar y tocar el **Estado Previo**, la **Condición** o el **Paso** (o cualquier combinación de ellos) hasta que quede afinado. Pero… Si las repeticiones deben ser de 2000 no pruebes las 2000 vueltas! Probá 3, si funciona para 3, funcionará para 2000.*

En Python tenemos a disposición El **Bucle for**. Este es un **Ciclo** que tiene automatizado, o incorporado al encabezado, el manejo del **Estado Previo** y del **Paso**. Por eso es tan popular.

Sintaxis de for

for *var_ctrl* **in** *iterable*:

-

-

-

Nota: Como iterable podremos usar cualquier dato estructurado, o una estructura de control o constructor como range()

¿Cómo se ejecuta **for** de Python?

Cuando el **FCP** llega a una sentencia **for** lo primero que se hace es verificar que en el **iterable** existan valores, si es así, se asigna el primer valor del mismo a la variable de control (**var_ctrl**) y se procede a ejecutar completamente el **Cuerpo**. Una vez finalizada la ejecución completa del **Cuerpo** el **FCP** vuelve al encabezado del **for** y asigna el valor siguiente en el **iterable** a la **variable de control**; repitiendo la ejecución del **Cuerpo**. Cuando se acaban los valores en el **iterable** y el **FCP** no dispone de más valores para asignarle a la **variable de control** el **Ciclo** finaliza y el **FCP** pasa a ejecutar la sentencia siguiente al **for** (la primera debajo, que estará alineada al encabezado del **for**, o aún más a su izquierda).
Imaginemos que el **iterable** es una bolsa con caramelos que sirven para alimentar la **variable de control**. En cada vuelta, la **variable de control** se come uno, con la energía que le aporta el dulce pega una vuelta completa al **Bucle** (una pasada al **Cuerpo**) y vuelve a intentar alimentarse. Al acabarse los caramelos en la bolsa se acaba la fuente de energía para la **variable de control** y el **for** finaliza.
Ojo! A no confundirse, aunque no queden valores para rellenar la **variable de control**, esto no significa que los valores hayan desaparecido del **iterable**, siguen allí. Pensemos que la **variable de control** tiene autorizada sólo una probadita por caramelo, pero no los retira de la bolsa.

Por ahora usaremos **rango** o **string** como opción para determinar los valores de la variable de control. Ya que son los únicos **iterables** que manejaremos por el momento.

Un **rango** es un constructor o estructura de control que permite generar una sucesión de números enteros, con un cierto paso o salto regular.
Sintaxis de rango
range([*<valor_inicio>*,]*<valor_fin>*[,*<paso>*])

Ejemplos:

range(1,10)	1,2,3,4,5,6,7,8,9
range(6)	0,1,2,3,4,5
range(0,8,2)	0,2,4,6
range(15,10,-5)	15

Tener en cuenta que:
El paso DEBE ser un entero
valor_inicio está incluído en el rango; valor_fin NO
valor_inicio es opcional; si no se coloca se asume 0
paso es opcional; si no se coloca se asume 1
Si se desea rango decreciente se debe colocar un paso negativo
Si se coloca paso DEBE COLOCARSE valor_inicio, aunque este último sea 0

Versión del programa de contabilidad de múltiplos de 5 con **for**:

```
# Contaremos cuántos múltiplos de 5 ingresan en un lote de 5 números
print('Ingresá 5 Números enteros')
totalMult=0
for vuelta in range(5):
    num=int(input('Número: '))
    if num%5==0:
        totalMult+=1
print('Vinieron:',totalMult,'múltiplos de 5')
```

Como ya vimos, un **Ciclo** repite un bloque de sentencias (al que llamamos **Cuerpo** del **Ciclo**). Por supuesto que ese **Cuerpo** puede estar conformado con cualquier combinación y cantidad de sentencias válidas de Python. Por lo tanto, resulta obvio suponer que dentro de un **Ciclo** pueda haber otro. Cuando se da esto, decimos que los **Ciclos** están **anidados**. El **FCP** de un programa ejecuta un **Bucle** hasta salir y recién sigue por la sentencia siguiente al mismo. Por lo tanto, cuando el **FCP** ingrese a un **Ciclo** interno (que está dentro de otro más externo), se quedará repitiendo este hasta finalizar y recién proseguirá ejecutando el **Cuerpo** del más externo.

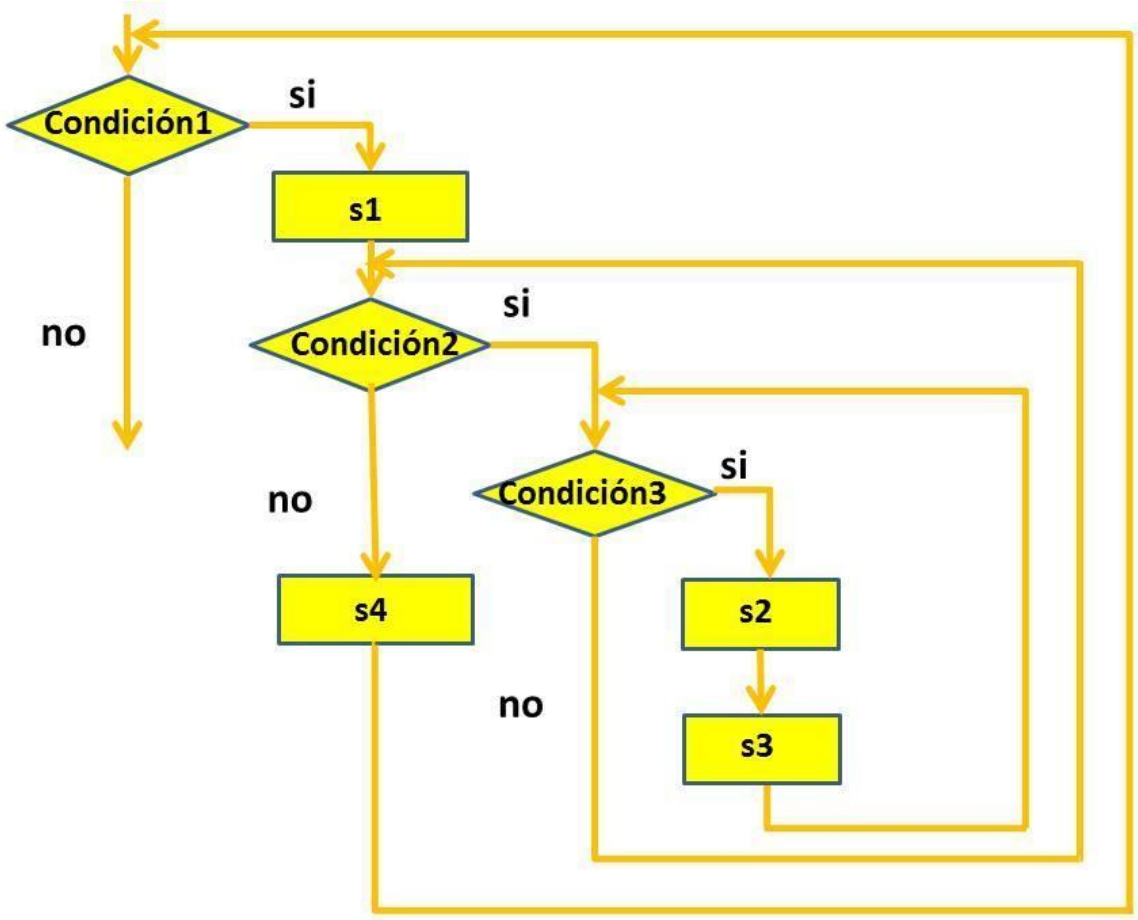
Por ello siempre los **Cuerpos** de **Bucles** internos se ejecutan más seguido que los de los externos. Las calesitas internas giran varias veces por cada vuelta de la externa.

¿Cuántos niveles de anidamiento se puede tener? Los que necesitemos!

Sólo debemos considerar:

- Cuando hay varios **Ciclos anidados** hay que tener mucho cuidado de no alterar dentro de uno interno la **variable de control** de un **Bucle** externo. Para ser ordenados y evitar problemas, cada **Bucle** con su **variable de control**. *Los relojes que emplean misma aguja para horas, minutos y segundos, no son muy prácticos.*

Este es un ejemplo gráfico de tres **Bucles anidados**:



En este ejemplo el **Bucle** de la **Condición 3** debe completarse antes de continuar ejecutándose el **Cuerpo** del **Bucle** de la **Condición 2**. Y lo propio debe ocurrir con éste, antes de proseguir con la ejecución del **Cuerpo** del **Bucle** de la **Condición 1**. Cada vuelta de un **Bucle** externo relanza completamente el interno. Por eso, si el **Ciclo** externo se ejecuta 3 veces, el del medio se ejecuta 5 veces por cada vuelta del externo, y el interno 10 vueltas por cada una del medio: El **Cuerpo 1** se ejecutará 3 veces, el **Cuerpo 2**, 15 y el **Cuerpo 3** 150 veces.

Usaremos una lógica inversa para plantear el empleo de Ciclos anidados. En primer término haremos un programa que cuente cuántos divisores tiene un numero natural (entero positivo).
Notas:

- **Todo número natural es divisible por si mismo y por la unidad, ergo, no los contamos.**
- **Un número dividido por si mismo produce cociente 1. El próximo cociente entero (2) lo obtenemos al dividir cualquier número por su mitad.**
- **De las dos consideraciones anteriores deducimos que la búsqueda de potenciales divisores de un número natural debería estar circunscripta al rango de enteros entre 2 y la mitad del número, inclusive.**

```
num=int(input('Ingresá un número entero positivo: '))
while num<=0:
    num=int(input('Ingresá un número entero positivo: '))
```

```
cantDiv=0
for d in range(2,num//2+1):
    if num%d==0:
        cantDiv+=1
print(num,'tiene',cantDiv,'divisores')
```

Algunas observaciones:

- **Para construir un rango puedo usar expresiones más complejas, no deben ser sólo constantes o variables. Lo único relevante es que el valor resultante sea entero.**
- **Usamos un bucle while para validar el ingreso y descartar números que no sean naturales. No basta con pedirle (o incluso implorarle) al usuario que haga algo, debemos asegurarnos que lo cumpla. La regla es la siguiente: nunca confíes en que el usuario obedezca; y dale todas las oportunidades que necesite para corregirse.**

Ahora averiguaremos la cantidad de divisores extras que tienen 10 números ingresados por el usuario. Pensemos: ya tenemos resuelto cómo contar la cantidad de divisores de un número en el programa de arriba. Si tenemos que hacer eso 10 veces en lugar de 1 lo que necesitamos es repetir todo el programa anterior 10 veces!

```
print('Ingresá 10 números y te contamos cuántos divisores tiene cada uno')
for n in range(10):
    num=int(input('Ingresá un número entero positivo: '))
    while num<=0:
        num=int(input('Ingresá un número entero positivo: '))
    cantDiv=0
    for d in range(2,num//2+1):
        if num%d==0:
            cantDiv+=1
    print(num,'tiene',cantDiv,'divisores')
```

Como observarás, ahora tenemos 3 **Bucles**; uno externo (**for** con variable de control **n**), y dos internos que son disjuntos entre sí (el **while** de validación y el **for** con variable de control **d**). Por cada vuelta del externo puede girar el **while** varias veces (eso dependerá de lo obediente que sea el usuario) y también girará algunas veces el **for** de búsqueda de divisores (eso dependerá del tamaño del número ingresado). Por último recalcaremos que si tenemos **anidamiento** de **Bucles for**, cada cual con su variable de control. ***Cada calesita con su sortija!***

Como ya hemos completado la incorporación de las herramientas algorítmicas más importantes, en el próximo apunte daremos un viraje importante. A partir de lo que viene haremos foco en los recursos que disponemos para hacer una organización inteligente de los datos.

Bibliografía Adicional

Wachenchauzer, R. (2018). *Aprendiendo a programar usando Python como herramienta*. Capítulo 5.