Programación 1

Tecnicatura Universitaria en Inteligencia Artificial

2022

Apunte 4

1. Construcciones Iterativas

Las computadoras están especialmente diseñadas para todas aquellas aplicaciones en las cuales una operación o conjunto de ellas deben repetirse muchas veces. Un tipo muy importante de estructura de programación es la construcción necesaria para repetir una o varias acciones una cantidad definida o indefinida de veces.

Podemos definir un bucle o ciclo como la sección de código que se repite y que se denomina así ya que cuando termina la ejecución de la última sentencia el flujo de control vuelve a la primera sentencia y comienza otra repetición de las sentencias del código. Cada repetición se conoce como iteración o pasada a través del bucle.

2. Ciclos definidos

Supongamos que se desea calcular un promedio de una lista de números escritos desde teclado —por ejemplo, las 5 calificaciones de un estudiante en un curso—. El medio conocido hasta ahora es leer los números y añadir sus valores a una variable `suma` que contenga las sucesivas sumas parciales. La variable `suma` se hace igual a cero y a continuación se incrementa en el valor del número cada vez que uno de ellos se lea. Un programa que resuelve este problema es:

```
suma = 0

nota = int(input("Ingrese una nota: "))
suma = suma + nota
nota = int(input("Ingrese una nota: "))
suma = suma + nota
nota = int(input("Ingrese una nota: "))
suma = suma + nota
nota = int(input("Ingrese una nota: "))
suma = suma + nota
nota = int(input("Ingrese una nota: "))
suma = suma + nota

print("El promedio es:", suma / 5)
```

Si se desease calcular el promedio para una mayor cantidad de notas, la esencia del programa no cambiará, solo basta repetir las siguientes acciones la cantidad de veces necesaria:

```
nota = int(input("Ingrese una nota: "))
suma = suma + nota
```

Estas acciones repetidas se denominan bucles o ciclos, mientras que la acción (o acciones) que se repite(n) en un bucle se denomina(n) iteración(es).

Para resolver este tipo de problema (repetir un cálculo para los valores contenidos en un intervalo dado) de manera eficiente, introducimos el concepto de ciclo definido, que tiene la siguiente forma:

```
for <variable> in <secuencia de valores>:
        <cuerpo>
```

Para el ejemplo introducido anteriormente,

```
suma = 0

for i in [1, 2, 3, 4, 5]:        # Cada una de las 5 calificaciones
        nota = int(input("Ingrese una nota: "))
        suma = suma + nota

print("El promedio es:", suma / 5)
```

La variable utilizada para recorrer la secuencia de valores se denomina variable de iteración y cambia para cada iteración del ciclo, controlando la ejecución y finalización del mismo.

Este ciclo se llama definido porque desde el comienzo se sabe exactamente cuántas veces se ejecutará el cuerpo ya que se define a partir de la secuencia de valores especificada.

La secuencia de valores puede definirse "a mano" escribiendolos entre corchetes. Por ejemplo,

```
for x in [1, 3, 9, 27]:
    print x * x
```

imprimirá los cuadrados de los números 1, 3, 9 y 27.

Nota

La variable de iteración puede ser utilizada luego en el cuerpo del ciclo, como muestra el ejemplo anterior.

2.1. range

En la aplicación de ciclos definidos será de suma importancia el uso de `range()` para generar una secuencia de números enteros. Por defecto, estos van desde 0 hasta el número que se pasa como parámetro menos 1.

```
range(5)        # Establece la secuencia de valores [0,1,2,3,4]
range(n)        # Establece la secuencia de valores [0, 1, ..., n-1].
```

De manera más general, `range()` retorna una lista de elementos, desde `inicio` (incluyendo este valor) hasta `fin` (sin incluir), en incrementos definidos por `salto`. El único argumento requerido es `fin`. Si `inicio` no se incluye, se asume que es 0; si `salto` no se incluye se asume 1.

```
# range(inicio, fin, salto)
```

```
range(11,16)     # Establece la secuencia de valores a [11, 12, 13, 14, 15]
range(n1, n2)  # Establece la secuencia de valores a [n1, n1+1, ..., n2-1]
range(1, 10, 2)  # Establece la secuencia de valores [1, 3, 5, 7, 9]
```

El problema analizado al finalizar unidad anterior decía:

Leer un número. Si el número es positivo escribir un mensaje "Número positivo", si el número es igual a 0 un mensaje Ïgual a 0", y si el número es negativo escribir un mensaje "Número negativo".

Se nos plantea a continuación un problema similar:

Problema

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

Utilizando los ciclos definidos, ahora es posible preguntarle al usuario cada vez, al inicio del programa, cuántos números va a ingresar para consultar. La solución propuesta resulta:

```
cant = input("Cuantos números quiere procesar: ")
for i in range(cant):
        x = float(input("Ingrese un número: "))
        if x > 0:
                print("Número positivo")
        elif x == 0:
                print("Igual a 0")
        else:
                print("Número negativo")
```

Su ejecución es exitosa:

```
>>> Cuantos números quiere procesar: 3
Ingrese un número: 25
Número positivo
Ingrese un número: 0
Igual a 0
Ingrese un número: -5
Número negativo
>>>
```

Sin embargo, es probable que considere que este programa no es muy intuitivo, porque lo obliga a contar de antemano cuántos números va a querer procesar, sin equivocarse, en lugar de ingresar uno a uno los números hasta procesarlos a todos.

3. Ciclos indefinidos

Para poder resolver este problema sin averiguar primero la cantidad de números a procesar, debemos introducir una instrucción que nos permita construir ciclos que no requieran que se informe de antemano la cantidad de veces que se repetirá el cálculo del cuerpo. Se trata de ciclos indefinidos en los cuales se repite el cálculo del cuerpo mientras una cierta condición es verdadera.

Un ciclo indefinido es de la forma

```
while <condición>:
        <hacer algo>
```

Donde `while` es una palabra reservada, y `<condición>` es una expresión booleana, igual que en las instrucciones `if`. El cuerpo (`<hacer algo>`) es, como siempre, una o más instrucciones.

El sentido de esta instrucción es el siguiente:

1. Evaluar la condición.
2. Si la condición es falsa, salir del ciclo.
3. Si la condición es verdadera, ejecutar el cuerpo.
4. Volver al inicio del ciclo y evaluar la condición (volver al punto 1).

4. Ciclo interactivo

¿Cuál es la condición y cuál es el cuerpo del ciclo en nuestro problema? Claramente, el cuerpo del ciclo es el ingreso de datos y la verificación de si es positivo, negativo o cero. En cuanto a la condición, es que haya más datos para seguir calculando.

Definimos una variable `hayMasDatos`, que valdrá `"Si"` mientras haya datos.

Se le debe preguntar al usuario, después de cada cálculo, si hay o no más datos. Cuando el usuario deje de responder `"Si"` , dejaremos de ejecutar el cuerpo del ciclo.

Una primera aproximación al código necesario para resolver este problema podría ser:

```
while hayMasDatos == "Si":
        x = float(input("Ingrese un número: "))
        if x > 0:
                print("Número positivo")
        elif x == 0:
                print("Igual a 0")
        else:
                print("Número negativo")

        hayMasDatos = input(" Quiere  seguir? (Si-No): ")
```

Sin embargo, si ejecutamos este programa tal como fue presentado nos encontraremos con un mensaje de error. El problema que se presentó en este caso, es que `hayMasDatos` no tiene un valor asignado en el momento de evaluar la condición del ciclo por primera vez.

Nota

Es importante prestar atención a cuáles son las variables que hay que inicializar antes de ejecutar un ciclo: al menos tiene que tener algún valor la expresión booleana que lo controla.

Una posibilidad es preguntarle al usuario, antes de evaluar la condición, si tiene datos; otra posibilidad es suponer que si ejecuta este programa es porque tiene algún dato para calcular, y darle el valor inicial `"Si"` a `hayMasDatos`.

Escribiendo el código necesario para implementar la segunda posibilidad se obtiene:

```
hayMasDatos = "Si"
while hayMasDatos == "Si":
        x = float(input("Ingrese un número: "))
        if x > 0:
```

```
        print("Número positivo")
        elif x == 0:
                print("Igual a 0")
        else:
                print("Número negativo")

        hayMasDatos = input(" Quiere  seguir? (Si-No): ")
```

El esquema del ciclo interactivo es el siguiente:

1. `hayMasDatos` hace referencia a `"Si"`.
2. Mientras `hayMasDatos` haga referencia a `"Si"`:
   a) Pedir datos
   b) Realizar cálculos
   c) Preguntar al usuario si hay más datos (`"Si"` cuando los hay). `hayMasDatos` hace referencia al valor ingresado.

Ésta es una ejecución del programa presentado:

```
>>> Ingrese un número: 25
Número positivo
 Quiere  seguir? (Si-No):"Si"
Ingrese un número: 0
Igual a 0
 Quiere  seguir? (Si-No): "Si"
Ingrese un número: -5
Número negativo
 Quiere  seguir? (Si-No): "No"
>>>
```

5. Ciclo con centinela

Un problema que tiene nuestra primera solución es que resulta poco amigable preguntarle al usuario después de cada cálculo si desea continuar. Para evitarlo, se puede usar el método del centinela: un valor distinguido que, si se lee, le indica al programa que el usuario desea salir del ciclo.

En este caso, podemos suponer que si ingresa el carácter `'*'` es una indicación de que desea terminar.

El esquema del ciclo interactivo es el siguiente:

1. Pedir datos.
2. Mientras el dato pedido no coincida con el centinela:
   a) Realizar cálculos
   b) Pedir datos

El programa resultante es el siguiente:

```
x = input("Ingrese un número (   *    para terminar): ")
```

```
        while x != "*":
            x = float(x)
                if x > 0:
                        print("Número positivo")
                elif x == 0:
                        print("Igual a 0")
                else:
                        print("Número negativo")

                x = input("Ingrese un número ('*' para terminar): ")
```

Y su ejecución:

```
 >>>Ingrese un número ('*' para terminar): 25
Número positivo
Ingrese un número ('*' para terminar): 0
Igual a 0
Ingrese un número ('*' para terminar): -5
Número negativo
Ingrese un número ('*' para terminar): '*'
>>>
```

6. Control de ciclos

El ciclo con centinela es muy claro pero tiene un problema: hay dos lugares (la primera línea del cuerpo y la última línea del ciclo) donde se ingresa el mismo dato. Si tuviéramos que realizar un cambio en el ingreso del dato (cambio de mensaje, por ejemplo) deberíamos estar atentos y hacer dos correcciones iguales. Sería preferible poder leer el dato x en un único punto del programa. A continuación, tratamos de diseñar una solución con esa restricción.

Es claro que en ese caso la lectura tiene que estar dentro del ciclo para poder leer más de un número, pero entonces la condición del ciclo no puede depender del valor leído, ni tampoco de valores calculados dentro del ciclo.

Pero un ciclo que no puede depender de valores leídos o calculados dentro de él será de la forma:

1. Repetir indefinidamente:

    a) Hacer algo

Y esto se traduce a Python como:

```
while True:
        <hacer algo>
```

Un ciclo cuya condición es `True` parece ser un ciclo infinito (o sea que nunca va a terminar). ¡Pero eso es gravísimo! ¡Nuestros programas tienen que terminar!

6.1. `break`

Afortunadamente hay una instrucción de Python, `break`, que nos permite salir de adentro de un ciclo (tanto sea `for` como `while` en medio de su ejecución.

En esta construcción:

```
while <condición>:
        <hacer algo_1>
        if <condif>:
                break
        <hacer algo_2>
```

El sentido de break es el siguiente:

1. Se evalúa y si es falsa se sale del ciclo.
2. Se ejecuta.
3. Se evalúa y si es verdadera se sale del ciclo (con break).
4. Se ejecuta.
5. Se vuelve al paso 1.

Diseñamos entonces:

1. Repetir indefinidamente
   a) Pedir dato.
   b) Si el dato ingresado es el centinela, salir del ciclo.
   c) Operar con el dato.

Codificamos en Python la solución al problema de los números usando ese esquema:

```
while True:
        x = input("Ingrese un número ('*' para terminar): ")
        if x == '*':
                break
        # Convertimos la cadena a numero
        x = float(x)
        if x > 0:
                print("Número positivo")
        elif x == 0:
                print("Igual a 0")
        else:
                print("Número negativo")
```

6.2. `continue`

En algunos casos dentro de un ciclo es conveniente finalizar la iteración actual y pasar a la siguiente. Para ello en Python existe la instrucción `continue`. El uso de `continue` al igual que el ya visto `break` se utiliza dentro de los ciclos y nos permite modificar su comportamiento.

Concretamente, `continue` se salta todo el código restante en la iteración actual y vuelve al principio en el caso de que aún queden iteraciones por completar. La diferencia entre el `break` y `continue` es que el continue no rompe el bucle, si no que pasa a la siguiente iteración saltando el código pendiente.

Un ejemplo de aplicación de esta sentencia podría utilizarse para resolver el problema anterior con la modificación que ahora no interesará mostrar ningún mensaje si el número ingresado es igual a 0.

```
while True:
        x = input("Ingrese un número ('*' para terminar): ")
        if x == '*':
                break
        x = float(x)
        if x > 0:
                print("Número positivo")
        elif x == 0:
                continue
        else:
                print("Número negativo")
```

6.3. pass

Ocasionalmente es útil tener el cuerpo de un ciclo (o de cualquier otra estructura) sin ninguna acción. Para esos casos, contamos con la instrucción `pass`. Cuya ejecución no produce efecto alguno más que evitar un error de sintaxis (ya que el intérprete espera que el cuerpo contenga "algo").

Usualmente se utiliza para reservar un espacio dentro del código para líneas que todavía no han sido escritas. Por ejemplo, si queremos resolver el problema anterior determinando números positivos y negativos, pero aún no hemos codificado que se hará en caso de que se ingrese un 0.

```
while True:
        x = input("Ingrese un número ('*' para terminar): ")
        if x == '*':
                break
        x = float(x)
        if x > 0:
                print("Número positivo")
        elif x == 0:
                pass
        else:
                print("Número negativo")
```

7. Bibliografía

- Apunte de la materia Algoritmos y Programación 1, primera materia de programación de las carreras de Ingeniería en Informática y Licenciatura en Análisis de Sistemas dela Facultad de Ingeniería de la UBA: http://materias.fi.uba.ar/7501/apunte%20PYTHON.pdf
- "Fundamentos de programación: Algoritmos, estructura de datos y objetos."(4ta Edición). Luis Joyanes Aguilar
- "Python for Everybody: Exploring Data Using Python 3". Charles R. Severance
- Automate the Boring Stuff with Python (2da Edición). Al Sweigart
- El Libro de Python. Libro digital:https://ellibrodepython.com/