Certified Tech Developer
The Ultimate Degree

Infraestructura I

Estructuras de control

Como hemos observado anteriormente, los algoritmos requieren dos estructuras de control importantes: iteración y selección. Ambas están disponibles en Python en varias formas. El programador puede elegir la instrucción que sea más útil para la circunstancia dada.

Para la iteración, Python proporciona una instrucción **while** estándar y una instrucción **for** muy potente. La instrucción **while** repite un cuerpo de código mientras una condición sea verdadera. Por ejemplo:

```
>>> contador = 1
>>> while contador <= 5:
...     print("Hola, mundo")
...     contador = contador + 1
```

```
Hola, mundo
Hola, mundo
Hola, mundo
Hola, mundo
Hola, mundo
```

Este código imprime la frase “Hola, mundo” cinco veces. La condición en la instrucción **while** se evalúa al inicio de cada repetición. Si la condición es **True**, el cuerpo de la instrucción se ejecutará. Es fácil ver la estructura de una instrucción **while** de Python debido al patrón de sangrado obligatorio que el lenguaje impone.

La instrucción **while** es una estructura iterativa de propósito general que usaremos en una serie de algoritmos diferentes. En muchos casos, una condición compuesta controlará la iteración. Un fragmento como el siguiente:

```
while contador <= 10 and not hecho:
...
```

Haría que el cuerpo de la instrucción fuera ejecutado solo en el caso en que se cumplan ambas partes de la condición. El valor de la variable **contador** tendría que ser menor o igual a 10 y el valor de la variable **hecho** tendría que ser **False** (**not False** es **True**) de modo que **True and True** da como resultado **True**.

Aunque este tipo de estructura es muy útil en una amplia variedad de situaciones, otra estructura iterativa, la instrucción **for**, se puede usar junto con muchas de las colecciones de Python. La instrucción **for** puede usarse para iterar sobre los miembros de una colección, siempre y cuando la colección sea una secuencia. Por ejemplo:

```
>>> for item in [1,3,6,2,5]:
...    print(item)
...
1
3
6
2
5
```

Asigna a la variable **item** cada valor sucesivo de la lista [1,3,6,2,5]. Entonces se ejecuta el cuerpo de la iteración. Esto funciona para cualquier colección que sea una secuencia (listas, tuplas y cadenas).

Un uso común de la instrucción **for** es implementar una iteración definida sobre un rango de valores. La siguiente instrucción:

```
>>> for item in range(5):
...    print(item**2)
...
0
1
4
9
16
```

```
>>>
```

Ejecutará la función **print** cinco veces. La función **range** devolverá un objeto de rango que representa la secuencia "0,1,2,3,4" y cada valor se asignará a la variable **item**. Este valor es entonces elevado al cuadrado y se imprime en pantalla.

Las instrucciones de selección les permiten a los programadores hacer preguntas y luego, con base en el resultado, realizar diferentes acciones. La mayoría de los lenguajes de programación proporcionan dos versiones de esta útil estructura: **ifelse** e **if**. Un ejemplo sencillo de una selección binaria que utiliza la instrucción **ifelse** es el siguiente:

```
if n<0:
  print("Lo siento, el valor es negativo")
else:
  print(math.sqrt(n))
```

En este ejemplo, el objeto referenciado por **n** se comprueba para ver si es menor que cero. Si es así, se imprime un mensaje indicando que es negativo. Si no es así, la instrucción ejecuta la cláusula **else** y calcula la raíz cuadrada.

Las estructuras de selección, como ocurre con cualquier otra estructura de control, pueden anidarse de modo que el resultado de una pregunta ayude a decidir si se pregunta la siguiente. Por ejemplo, supongamos que **puntaje** es una variable que contiene una referencia a un puntaje de un examen de ciencias de la computación.

```
if puntaje >= 90:
  print('A')
else:
  if puntaje >= 80:
        print('B')
  else:
        if puntaje >= 70:
        print('C')
        else:
        if puntaje >= 60:
        print('D')
        else:
        print('F')
```

Este fragmento clasificará un valor llamado **puntaje** mediante la impresión de la calificación cualitativa obtenida. Si el puntaje es mayor o igual a 90, la instrucción

imprimirá "**A**". Si no lo es (**else**), se hace la pregunta siguiente. Si el puntaje es mayor o igual a 80, entonces debe estar entre 80 y 89, ya que la respuesta a la primera pregunta era falsa. En este caso se imprimirá la letra "**B**". Puede verse que el patrón de sangrado de Python ayuda a dar sentido a la asociación entre **if** y **else** sin necesidad de elementos sintácticos adicionales.

Una sintaxis alternativa para este tipo de selección anidada utiliza la palabra clave **elif**. El **else** y el **if** siguiente se combinan para eliminar la necesidad de niveles de anidamiento adicionales. Tené en cuenta que el último **else** sigue siendo necesario para proporcionar el caso por defecto, en caso de que todas las demás condiciones fallen.

```
if puntaje >= 90:
  print('A')
elif puntaje >= 80:
  print('B')
elif puntaje >= 70:
  print('C')
elif puntaje >= 60:
  print('D')
else:
  print('F')
```

Python también tiene una estructura de selección de una sola vía, la instrucción **if**. Con esta instrucción, si la condición es verdadera, se realiza una acción. En caso de que la condición sea falsa, el procesamiento simplemente continúa con la instrucción que siga después del **if**. Por ejemplo, el siguiente fragmento comprobará primero si el valor de una variable **n** es negativo. Si lo es, entonces es modificado mediante la función de valor absoluto. Sea cual sea el caso, la siguiente acción es calcular la raíz cuadrada.

```
if n<0:
  n = abs(n)
print(math.sqrt(n))
```

Regresando a las listas, existe un método alternativo para crear una lista que usa estructuras de iteración y de selección, conocido como **comprensión de listas**. Una comprensión de lista permite crear fácilmente una lista basada en algunos criterios de procesamiento o de selección. Por ejemplo, si quisiéramos crear una lista de los primeros 10 cuadrados perfectos, podríamos usar una instrucción **for**:

```
>>> listaCuadrados=[]
>>> for x in range(1,11):
        listaCuadrados.append(x*x)

>>> listaCuadrados
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
```

Usando comprensión de listas, podemos hacer lo mismo en un único paso como

```
>>> listaCuadrados=[x*x for x in range(1,11)]
>>> listaCuadrados
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
```

La variable **x** toma los valores de 1 a 10 especificados por la estructura **for**. El valor de **x*x** se calcula y se agrega a la lista que se está construyendo. La sintaxis general para una comprensión de listas también permite agregar un criterio de selección para que únicamente se agreguen ciertos ítems. Por ejemplo:

```
>>> listaCuadrados=[x*x for x in range(1,11) if x%2 != 0]
>>> listaCuadrados
[1, 9, 25, 49, 81]
>>>
```

Esta comprensión de listas construyó una lista que contenía solamente los cuadrados de los números impares en el rango de 1 a 10. Cualquier secuencia que soporte la iteración se puede utilizar dentro de una comprensión de listas para construir una nueva lista.

```
>>>[letra.upper() for letra in 'estructuras' if letra not in 'aeiou']
['S', 'T', 'R', 'C', 'T', 'R', 'S']
>>>
```