El futuro digital es de todos

MinTIC

Bucle for y cadenas de string

OPERADO POR:

Universidad Tecnológica de Pereira

Mision TIC 2022

RUTA DE APRENDIZAJE 1

Bucle for

Los comandos que revisamos anteriormente son comandos lógicos, nos permiten indicarle a la maquina que un valor debe cumplir una condición particular para realizar alguna tarea. Ahora estos comandos no pueden realizar este proceso solos, necesitan de la ayuda de los comandos especiales:

IF

Es la palabra reservada para obtener el poder de los condicionales en el código.

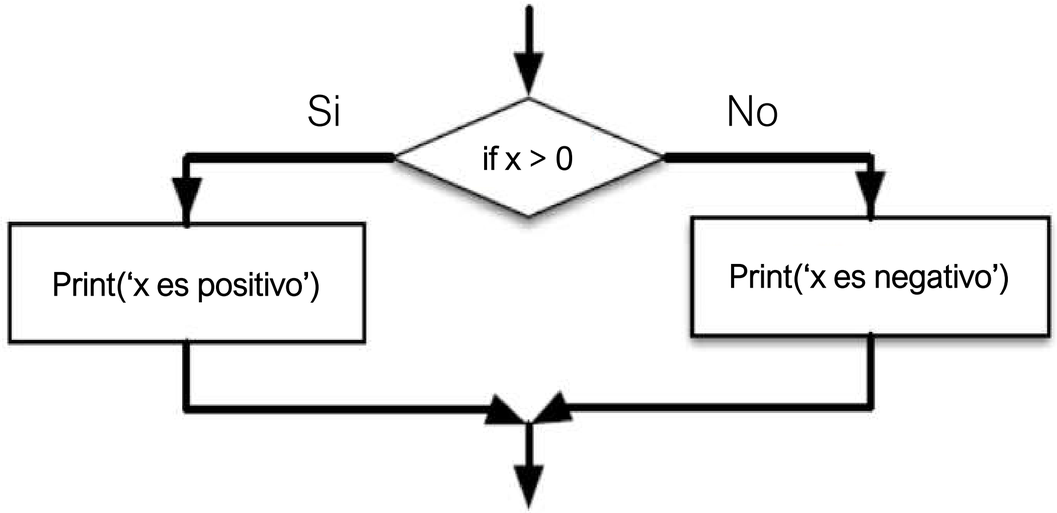
IF indica específicamente “si”, si se cumple una condición particular.

ELSE

Expresa “en el caso contrario”. Siguiendo con el ejemplo anterior de la lluvia: **if(va a llover)** coge el paraguas **else** dejarlo en casa.

Ejemplo:

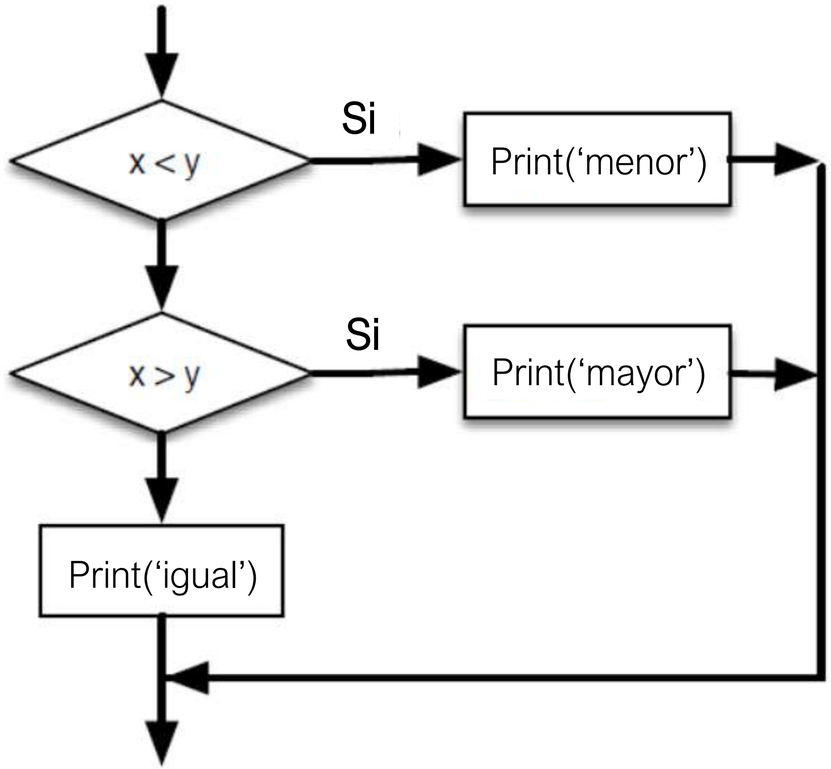
```
x = 10
if x > 0 :
    print('x es positivo')
else :
    print('x es negativo')
```



A veces hay más de dos posibilidades y necesitamos más de dos ramas. Una forma de expresar un cálculo como ese es un condicional encadenado:

```
x=
y=

if x < y:
    print('"x" es menor que "y"')
elif x > y:
    print('"x" is mayor que "y"')
else:
    print('"x" y "y" sons iguales')
```



Cada condición se comprueba en orden. Si la primera es falsa, la siguiente se comprueba, y así sucesivamente. Si una de ellas es verdadera, la rama correspondiente se ejecuta, y la declaración termina.

Incluso si más de una condición es cierta, sólo la primera rama verdadera se ejecuta.

```
letra =

if letra == 'a':
    print('Malas resultado')
elif letra == 'b':
    print('Buen resultado ')
elif letra == 'c':
    print('Cerca, pero no es correcto')
```

Condicionales anidadas:

Un condicional también puede anidarse dentro de otro. Podríamos haber escrito el ejemplo de las tres ramas así:

```
if x == y:
    print('x y y son iguales')
else:
    if x < y:
        print('x es menos que y')
    else:
        print('x es mayor que y')
```

El condicional exterior contiene dos ramas. La primera rama contiene una declaración simple. La segunda rama contiene otra declaración if, que tiene dos ramas propias. Esas dos ramas son ambas declaraciones simples, aunque también podrían haber sido declaraciones condicionales.

Aunque la indentación de las declaraciones hace que la estructura sea aparente, los condicionales anidados se vuelven difíciles de leer muy rápidamente. En general, es una buena idea evitarlos cuando se pueda.

Los operadores lógicos suelen ser una forma de simplificar las declaraciones condicionales anidadas. Por ejemplo, podemos reescribir el siguiente código utilizando un solo condicional:

```
if 0 < x:
    if x < 10:
        print('x es un número positivo de un solo dígito.')
```

La declaración print se ejecuta sólo si pasamos los dos condicionales, para que podamos obtener el mismo efecto con el operador y:

```
if 0 < x and x < 10:
    print('es un número positivo de un solo dígito')
```

Estructura Cíclicas:

Con la instrucción if, podemos generar muchas condiciones especiales, sin embargo necesitamos un conjunto de instrucciones que se ejecute varias veces dependiendo de alguna condición o de los datos que proporcione el usuario.

Hasta el momento los programas que hemos construido ejecutan cada instrucción una única vez, a menos que se encuentren dentro de funciones que se llamen varias veces. El problema es que desde que se escriba el programa también va a quedar establecida la cantidad de veces que se llame cada función.

Este problema puede solucionarse a través del uso de **instrucciones iterativas**, las cuales nos permiten expresar cuántas veces tiene que ejecutarse una instrucción sin tener que escribirla muchas veces.

En esta sección vamos a introducir el concepto de instrucciones iterativas y vamos a explicar cómo se implementan en Python usaremos las instrucciones **while y for.**

FOR

Con esta estructura se puede hacer cualquier cosa que nosotros necesitemos. Literalmente, todo lo que puedas hacer con los demás *bucles* es posible de hacer con este ciclo.

Cada uno de los siguientes elementos solitarios del bucle también debe tener una sangría de 4 o más espacios.

Si una línea no tiene sangría, se considera que está fuera del bucle y también terminará cualquier línea adicional considerada en el bucle. Un error común es eliminar los espacios y, por lo tanto, finalizar el ciclo antes de tiempo.

En bucle **for** se utiliza para ejecutar un bloque de instrucciones un número determinado de veces.

Para los bucles **for** se utiliza una variable de contador cuyo valor aumente o disminuya con cada repetición del bucle.

Ritmo de iteración: básicamente, es el ritmo al que se consume el *bucle*. Si, por ejemplo, escribimos ***x=x+2***, el *bucle* itera de 2 en 2, aunque lo más común es que se haga de 1 en 1 con la siguiente sintaxis ***x++***.

```
for x=0;x<10;x++
```

WHILE

El bucle **while** es más sencillo de entender. Si has entendido el bucle *for*, este también será fácil. Básicamente, esta es una estructura iterativa a la que solo hay que pasarle una condición de parada: *while(condicion de parada)*.

Ejemplo:

Hace que un procedimiento se ejecute **4 veces**. La instrucción **for** especifica la variable de contador x, y sus valores inicial y final. Python incrementará automáticamente la variable contador (x) en 1 después de llegar al final del bloque.

```
for x in range(0, 3):
    print("Estamos en la iteración " + str(x))
```

Python puede usar cualquier método iterable como contador de bucle **for**. En el caso anterior estamos usando **range()**. Otros objetos iterables pueden ser listas o **string**. También puede crear sus propios objetos iterables si es necesario.

A veces es necesario aumentar o disminuir la variable de contador según el valor que especifique. En el siguiente ejemplo, la variable del contador j se incrementa en 2 cada vez que se repite el ciclo. Cuando finaliza el ciclo, el total es la suma de 0, 2, 4, 6 y 8:

```
for j in range(0, 10, 2):
    print("Estamos en la iteración " + str(j))
```

Puede salir de cualquier instrucción **for** antes de que el contador alcance su valor final mediante la instrucción **break**. Dado que normalmente solo desea salir en determinadas situaciones, como cuando se produce un error. También se puede utilizar la instrucción **if** en el bloque de instrucción **True** . Si la condición es Falsa , el ciclo se ejecuta como de costumbre.

```
oracion = 'Pablo entiende muy bien Python'
frases = oracion.split() # convierte a una lista cada palabra
print("La oración analizada es:", oracion, ".\n")

for palabra in range(len(frases)):
    print("Palabra: {0}, en la frase su posición es: {1}".format(
        frases[palabra], palabra))
```

```
La oración analizada es: Pablo entiende muy bien Python .

Palabra: Pablo, en la frase su posición es: 0
Palabra: entiende, en la frase su posición es: 1
Palabra: muy, en la frase su posición es: 2
Palabra: bien, en la frase su posición es: 3
Palabra: Python, en la frase su posición es: 4
```

A pesar de que el comando break puede llegar a ser muy útil, no es una buena practica de programación su uso constante.

Bucle ‘for’ con ‘else’

Al igual que la sentencia if y el bucle **while**, la estructura **for** también puede combinarse con una sentencia else.

El nombre de la sentencia else es equivocada, ya que el bloque else se ejecutará en todos los casos, es decir, cuando la expresión condicional del bucle for sea False, (a comparación de la sentencia if).

```
db_connection = "127.0.0.1","5432","root","nomina"

for parametro in db_connection:
    print(parametro)
else:
    print("""El comando PostgreSQL es:
$ psql -h {server} -p {port} -U {user} -d {db_name}""".format(
        server=db_connection[0], port=db_connection[1],
        user=db_connection[2], db_name=db_connection[3]))
```

Manejo de cadenas de string

String

En clases anteriores trabajamos con cadenas de string para entregar resultados estructurados simples sin embargo aun debemos hacer mas énfasis en todo lo que es posible realizar al usar strings.

Saber cómo manipular cadenas de caracteres juega un papel fundamental en la mayoría de las tareas de procesamiento de texto. Si quieres experimentar con las siguientes lecciones puedes escribir y ejecutar pequeños programas tal como lo hicimos en clases previas, o puedes abrir tu intérprete de comandos de Python / Terminal para probarlos ahí.

Una string es una secuencia de caracteres. Puede acceder a los caracteres de uno en uno con el operador de paréntesis:

La segunda afirmación extrae el carácter en la posición 1 del índice de la variable de la fruta y lo asigna a la variable de la letra. La expresión entre paréntesis se llama índice. El índice indica el carácter de la secuencia que se desea (de ahí el nombre). Pero puede que no consigas lo que esperas:

```
fruta = 'fresa'
```

```
letra = fruta[1]
```

```
print(letra)
```

```
r
```

Para la mayoría de la gente, la primera letra de "banana" es "b", no "a". Pero en Python, el índice es un desplazamiento del principio de la cadena, y el desplazamiento de la primera letra es cero (0).

```
letra = fruta[0]
print(letra)
```

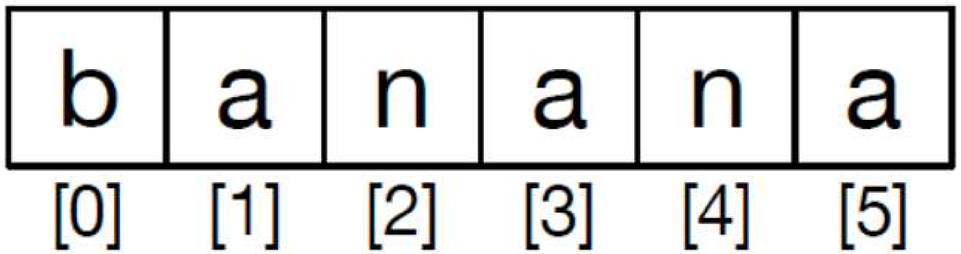
```
b
```

Así que "b" es la letra en la posición 0 de la palabra "banana", "a" es la letra en la posición 1, y "n" seria la letra en la posición 2.

Se puede usar cualquier expresión, incluyendo variables y operadores, como un índice, pero el valor del índice tiene que ser un número entero. De lo contrario, se obtiene:

```
letra = fruta[1.5]
```

Recordemos que cada letra tiene su espacio en memoria:



Obtener la longitud de un string usando len():

len es una función incorporada que devuelve el número de caracteres de una cadena:_

```
In [87]: fruta = 'banana'
         len(fruta)

Out[87]: 6
```

Para obtener la última letra de un string, podrías estar tentado de intentar algo como esto:

```
longitud = len(fruta)
ultimo = fruta[longitud]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-7-deffcf8ac8c9> in <module>
      1 longitud = len(fruta)
----> 2 ultimo = fruta[longitud]

IndexError: string index out of range
```

La razón del **IndexError** es que no hay ninguna letra en "banana" con el índice 6. Desde que empezamos a contar en el cero, las seis letras están numeradas del 0 al 5.

Para obtener el último carácter, hay que restarle 1 a la longitud:

```
ultimo = fruta[longitud-1]
print(ultimo)
```

```
a
```

Alternativamente, puedes usar índices negativos, que cuentan hacia atrás desde el final de la cuerda. La expresión fruta[-1] da la última letra, fruta[-2] da la el penúltimo, y así sucesivamente.

Rebanas de String

Un segmento de un string se llama un trozo. Seleccionar una rebanada es similar a seleccionar una caracter:

```
In [90]: s = 'Monty Python'
         print(s[0:5])

Monty
```

```
In [91]: print(s[6:12])

Python
```

El operador devuelve la parte de la cadena del "n-ésimo" carácter "m-ésimo" al carácter, incluyendo el primero pero excluyendo el último [n,m). Si se omite el primer índice (antes de los dos puntos), el trozo comienza al principio de la cadena.

Si se omite el segundo índice, la rebanada va al final de la cadena:

```
In [92]: fruta = 'banana'
         fruta[:3]

Out[92]: 'ban'

In [93]: fruta[3:]

Out[93]: 'ana'
```

Si el primer índice es mayor o igual que el segundo el resultado es una cadena vacía, representada por dos comillas:

```
In [94]: fruta = 'banana'
         fruta[3:3]

Out[94]: ''
```

El futuro digital es de todos

MinTIC

GRACIAS

OPERADO POR:

Universidad Tecnológica de Pereira

MisiónTIC2022