OPERADORES

ARITMÉTICOS

Operador	Descripción	Ejemplo
+	Suma	`>>> 5+5` `10`
-	Resta	`>>> 5-5` `0`
/	División	`>>> 6/5` `1.2`
//	División Entera	`>>> 6//5` `1`
*	Multiplicación	`>>>5*5` `25`
**	Potenciación	`>>>5**5` `3125`
%	Módulo	`>>>6%5` `1`

LÓGICOS

Operador	Alterno	Descripción	Ejemplo
&	and	Operación and	`>>> True and False` `False`
|	or	Operación or	`>>> True | False` `True`
^		Operación xor	`>>> True ^ False` `True`
~	not	Negación	`>>> not(True)` `False`
<<		Desplazar bit a izquierda	`>>> 4<<1` `8`
>>		Desplazar bit a derecha	`>>> 4>>1` `2`

RELACIONALES

Operador	Descripción	Ejemplo
a==b	a Igual que b?	`>>> 5==5` `True`
a!=b	a diferente de b?	`>>> 5!=5` `False`
a>b	A mayor que b?	`>>> 6>5` `True`
a<b	A menor que b?	`>>> 6<5` `False`
a>=b	A mayor o igual que b.?	`>>> 6>=5` `True`
a<=b	A menor igual que b?	`>>> 6<=5` `False`

ESTRUCTURAS DE CONTROL DE FLUJO

El control de flujo se refiere al orden en el que se ejecutan las instrucciones contenidas en un programa. Cuando un programa tiene más de una posibilidad de flujo es necesario utilizar estructuras de control que decidan qué camino tomar en la ejecución a partir del cumplimiento o no de ciertas condiciones. continuación se expone los tipos de estructuras de control de flujo que existen en Python.

SENTENCIAS CONDICIONALES

Las sentencias condicionales permiten la decisión entre la ejecución o no de bloques de código a partir del cumplimiento de una condición. Esta condición se evalúa mediante operadores relacionales que retornan un dato booleano. A continuación se expone las sentencias de control de flujo básicas de Python.

IF

Esta es la sentencia de control de flujo más básica. Permite o no la ejecución de un bloque de código a partir de la verificación de una condición. Hay que recordar que Python es un lenguaje identado, esto quiere decir que el bloque de código asociado al *if* debe ir tabulado. A continuación **se exponen algunos ejemplos de**l uso de este condicional.

```
x=6
y=12
z=True
if (y>8):
        print ("Y mayor que 8")
if (x>8):
        print ("X mayor que 8")
if (z):
        print ("Z verdadero")
print ("Fin del programa")
```

El resultado de la ejecución de este código es el siguiente:

```
Y mayor que 8
Z verdadero
Fin del programa
```

También es posible verificar más de una condición utilizando conectores lógicos. Por ejemplo:

```
x=6
y=12
if (y>8 and x>8):
        print("Se cumplen las dos condiciones")
if (y>8 or x>8):
        print ("Se cumple una o las dos condiciones")
```

El resultado de la ejecución del anterior código es:

```
Se cumple una o las dos condiciones
```

IF ... ELSE

La sentencia *else* va luego de una sentencia *if* y tiene como objetivo que el bloque de código asociado a esta se ejecute únicamente si la condición expresada en el *if* no se cumple. Una traducción de esta sentencia es "Si se cumple la condición se debe ejecutar un bloque de código, sino se cumple se debe ejecutar otro bloque de código". Acá un ejemplo de esta sentencia:

```
x=5
y=6
if (x>y):
        print ("x mayor que y")
else:
        print ("x no es mayor que y")
```

IF . .. ELIF

La sentencia *elif* tiene que ir después de de una sentencia *if*. Esta hace que en el caso de que no se cumpla la condición del if se verifique una segunda condición. Una traducción a esta sentencia es "Si se cumple la condición 1 se debe ejecutar un bloque de código, si no se cumple la condición 1 se verifica una condición 2 y si esta se cumple se ejecuta otro bloque de código". Acá un ejemplo de esta sentencia:

```
x=3.5
if (x>4):
        print ("Rendimiento excelente")
elif (x>3):
        print ("Rendimiento aceptable")
else:
        print ("Reprobado")
```

SENTENCIAS CÍCLICAS

Las sentencias cíclicas permiten la repetición de un bloque de código un número de veces determinado por una condición o por arreglo de datos.

WHILE

Esta sentencia repite un bloque de código mientras se cumpla una condición. A continuación algunos ejemplos:

```
i=1
while i<4:
        print("Vuelta No: ",i)
        i=i+1
```

En el código anterior se ejecuta un bloque de código mientras que la variable *i* sea menor a 4. Cómo *i* comienza en 1 y se va incrementando de uno en uno, el bloque de código se ejecuta 3 veces.

```
m='n'
while m!='s':
        m=input("Desea salir (s/n): ")
```

En el código anterior se repite un bloque de código mientras que el usuario digite una letra diferente a 's'.

```
n=True
while n:
        print("Ciclo infinito")
```

En el código anterior se realiza un ciclo infinito. El ciclo se define de tal forma que se repita un bloque de código mientras *n*, cómo la variable *n* contiene el valor boolenao *True*, la condición siempre se va a cumplir y el ciclo siempre se va a repetir.

FOR

Esta sentencia permite repetir un bloque de código en el cual una variable va tomando para cada ciclo, uno a uno los valores definidos en un arreglo de datos. A continuación se presenta un ejemplo de *for* que utiliza una lista para definir los elementos que va a ir tomando la variable:

```
vocales = ['a','e','i','o','u']
for i in vocales:
        print(i)
```

En el siguiente ejemplo se utiliza una cadena de caracteres para definir los elementos que va a ir tomando la variable:

```
cadena = "hola"
for i in cadena:
        print(i)
```

Cuando los valores que debe tomar la variable asociada al *for* son numéricos secuenciales se recomienda utilizar la función *range*. A continuación se muestra un ejemplo de un *for* que utiliza la función *range*.

```
for i in range(4):
        print(i)
```

La función *range* también se puede definir de otras formas de tal manera que se indique el inicio, el fin, el paso incremental o decremental. En el siguiente ejemplo se ilustran varias formas de definir la función *range*. Note que para desplegarlo en pantalla el rango se convierte a una lista.

```
list(range(3))#el inicio y el paso se toman por defecto
list(range(3,5)) #el paso se toma por defecto = 1
list(range(4,10,2)) #el paso es 2
list(range(10,4,-2)) #el paso es -2
```

SENTENCIAS BREAK Y CONTINUE

Estas sentencias permiten alterar el flujo de ejecución de un bucle.

BREAK

La sentencia *Break* permite interrumpir la ejecución de un bucle. A continuación se ilustra su uso mediante un ejemplo que permite llenar una lista hasta con 100 valores, sin embargo el usuario puede interrumpir el llenado y dejar la lista con menos valores.

```
x=[]
for i in range(100):
        print("Posición ",i)
        a=input("Valor : ")
        x.append(a)
        m=input("Desea continuar ingresando (s/n):")
        if (m=='n'):
                break
```

CONTINUE

La sentencia *continue* permite omitir una porción de código de un bucle. Cuando esta sentencia se ejecuta, el programa pasa al siguiente ciclo, omitiendo el fragmento de código que faltaba por ejecutar dentro del bucle. A continuación se presenta un ejemplo donde se utiliza la sentencia *continue* para sacar un aviso en pantalla únicamente para los 3 primeros ciclos (en los cuales no se ejecuta la sentencia *continue*).

```
for i in range(5):
        print("Vuelta No. : ",i)
        if (i>2):
        continue
        print("En esta vuelta no se ejecutó la sentencia continue")
```

SENTENCIA ELSE EN BUCLES

La sentencia *else* en *Python* no es exclusivamente un complemento de la sentencia *if*, también puede utilizarse junto a una sentencia cíclica *for* o *while*. La sentencia *else* asociada a una sentencia cíclica permite la ejecución de un bloque de código únicamente cuando el bucle ha completado los ciclos completamente según la condición (para *while*) o el arreglo de datos (para *for*), si el bucle se ha interrumpido mediante un *break* el bloque de código asociado al *else* no será ejecutado.

TALLER

Utilizando la sentencia ELSE en bucles, desarrollar un juego que genere un número entero aleatorio de 0 al 99 y le pida al usuario que lo adivine. El número de intentos debe ser de 10. Al finalizar (adivine o no) se debe indicar si el usuario ganó o perdió y se le debe dar un puntaje.