TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE

- Operadores lógicos
- Operadores de comparación
- Comparaciones entre tipos
- Conversiones implícitas
- Crear un objeto Python
- Manipular los atributos de un objeto Python
- Sentencias básicas
- La instrucción if
- La instrucción for
- La instrucción while
- Las sentencias break y continue

OPERADORES LÓGICOS

Se utilizan para tomar una decisión basada en múltiples condiciones. Los operadores lógicos utilizados en Python son: and (y), or (o) y not (no).

El resultado de una operación lógica es *true* o *false*. True cuando la sentencia es verdadera, y false cuando esta es falsa.

And devuelve true cuando ambos elementos son verdaderos, de lo contrario devuelve falso.

Or devuelve true cuando alguno de los elementos es verdadero, de lo contrario devuelve falso.

Not devuelve true cuando la sentencia es falsa, o false cuando la sentencia es verdadera. Es decir, devuelve el resultado contrario a la sentencia.

```
1 x = True
2 y = False
3
4 print('x and y es',x and y)
5
6
```

```
#Resultado:
x and y es False
```

```
print('x or y es',x or y)

#Resultado:
x or y es True
```

```
print('not x es',not x)

#Resultado:
not x es False
```

OPERADORES DE COMPARACIÓN

Se utilizan para comparar valores. Devuelve True o False, según la condición.

- Operador mayor que (>), regresa true si el valor de la izquierda es mayor que el de la derecha.
- Operador menor que (<), regresa true si el valor de la izquierda es menor que el de la derecha.
- Operador igual que (==), regresa true si el valor de la izquierda es igual que el de la derecha.
- Operador no es igual que (!=), regresa true si el valor de la izquierda es diferente al de la derecha.
- Operador mayor o igual que (>=), regresa true si el valor de la izquierda es mayor o igual que el de la derecha.
- Operador menor o igual que (<=), regresa true si el valor de la izquierda es menor o igual que el de la derecha.

```
x = 10
y = 12

```

```
print('x > y es',x>y)

#Resultado:
x > y es False
```

```
print('x < y es',x<y)

#Resultado:
x < y es True
```

```
print('x == y es',x==y)

#Resultado:
x == y es False
```

```
print('x != y es',x!=y)

#Resultado:
x != y es True
```

```
print('x >= y es',x>=y)

#Resultado:
x >= y es False
```

```
print('x <= y es',x<=y)

#Resultado:
x <= y es True
```

COMPARACIONES ENTRE TIPOS

La función `type()` devuelve el tipo del objeto, o un nuevo objeto de tipo basado en los argumentos pasados.

CONVERSIONES IMPLÍCITAS

Son acciones realizadas por Python, en las cuales se cambia un tipo de dato por otro. Se ejecutan de forma automática al realizar ciertas operaciones, y no requieren una orden específica que indique que se desea hacer el cambio.

Un ejemplo en el cual podemos ver un tipo de conversión implícita es el caso en que tenemos 2 variables, una de tipo entero llamada “a”, y otra de tipo flotante llamada “b”. Si las sumamos, y asignamos el resultado a la variable “a”, ésta automáticamente cambiará su tipo a flotante para poder hacer la operación.

```
a = 1
b = 2.14
a = a + b
print(type(a))

#Resultado:
<class 'float'>
```

Sin embargo, hay casos donde Python no podrá hacer la conversión y el resultado será un error.

LA INSTRUCCIÓN IF

La toma de decisiones se realiza con la sentencia condicional **if**. Esta sentencia estudia si una expresión es verdadera o falsa; en caso de ser verdadera ejecuta una acción, y en caso contrario realiza otra. También podemos estudiar múltiples opciones. Para ello tenemos las siguientes expresiones:

if (si condicional): esta sentencia estudia si se cumple la condición que le sigue; devuelve true en caso de que se cumpla, y false en caso contrario. La sintaxis es como sigue:

```
if expresión a probar:
    instrucciones en caso de ser verdadero
```

Nótese que es importante la indentación, todas las instrucciones en caso de ser verdadero deben tener una indentación de al menos 4 espacios en blanco con respecto a la expresión **if**.

else (en caso contrario): esta sentencia siempre se usa para indicar qué debe suceder en caso de que la sentencia **if** sea falsa, siempre y cuando no existan más condiciones posibles. Es decir, solo tenemos 2 opciones. La sintaxis es como sigue:

```
if expresión a probar:
    Instrucciones en caso de ser verdadero
else:
    Instrucciones en caso de ser falso
```

De nuevo es importante tomar en cuenta la indentación, tanto para el **if**, como para el **else**.

elif (en caso contrario sí): esta sentencia se usa para indicar que hay que evaluar otra condición en caso de que el **if** tenga resultado falso. De esta manera podemos crear una cascada de expresiones a evaluar. Su sintaxis es como sigue:

```
if expresión a probar:
    Instrucciones en caso de ser verdadero
elif Segunda expresión a probar:
    Instrucciones en caso de ser verdadero
else:
    Instrucciones en caso de ser falso
```

De nuevo es importante tomar en cuenta la indentación tanto para el **if**, como para el **else** y el **elif**. Veamos un ejemplo: "'En este programa, comprobamos si el número es positivo o negativo o cero, y mostrar un mensaje apropiado"'

```
num = 3.4

if num > 0:
    print("Número positivo")
elif num == 0 :
    print("Cero")
else :
    print("Número negativo")
```

LA INSTRUCCIÓN FOR

El bucle **for** en Python se utiliza para iterar sobre una secuencia (lista, tupla, cadena), u otros objetos iterables. La iteración sobre una secuencia se denomina recorrido.

La sintaxis de la instrucción es:

```
for elem in secuencia:
    instrucciones del bucle
```

Aquí, **elem** es la variable que toma el valor del elemento dentro de la secuencia en cada iteración. La sintaxis incluye in, lo cual indica que son todos los valores dentro de la secuencia.

El bucle continúa hasta llegar al último elemento de la secuencia. El cuerpo del bucle **for** (instrucciones del bucle) se separa del resto del código utilizando indentación.

Un bucle **for** puede tener también un bloque **else** opcional. La parte **else** se ejecuta al agotarse los elementos de la secuencia en el bucle **for**.

LA INSTRUCCIÓN WHILE

El bucle **while** en Python se utiliza para iterar sobre un bloque de código mientras la expresión de prueba (condición) sea verdadera.

Generalmente, utilizamos este bucle cuando no sabemos de antemano el número de veces que hay que iterar.

Sintaxis del bucle **while**:

```
while expresión_de_prueba:
    Cuerpo del while
```

En el bucle **while**, la **expresión_de_prueba** se comprueba primero. Cuerpo del **while** se correrá sólo si la **expresión_de_prueba** se evalúa como **True**. Después de una iteración, la expresión de prueba se comprueba de nuevo. Este proceso continúa hasta que la **expresión_de_prueba** sea falsa.

En Python, el cuerpo del bucle **while** se determina mediante la indentación. El cuerpo comienza con sangría, y la primera línea sin sangría marca el final.

Python interpreta cualquier valor distinto de cero como **True**. Ninguno y 0 se interpretan como Falso.

Al igual que los bucles **for**, los bucles **while** también pueden tener un bloque **else** opcional. La parte **else** se ejecuta si la condición del bucle **while** se evalúa como **False**.

LAS SENTENCIAS BREAK Y CONTINUE

En Python, las sentencias ***break*** y ***continue*** pueden alterar el flujo de un bucle normal. Los bucles iteran sobre un bloque de código hasta que la expresión de prueba es falsa, pero algunas veces deseamos terminar la iteración actual, o incluso todo el bucle sin comprobar la expresión de prueba. Las sentencias **break** y **continue** se utilizan en estos casos.

La sentencia **break** termina el bucle que la contiene. El control del programa fluye hacia la sentencia inmediatamente posterior al cuerpo del bucle.

Si la sentencia **break** está dentro de un bucle anidado (bucle dentro de otro bucle), la sentencia **break** terminará el bucle más interno.

Sintaxis de break:

```
Break
```

La sentencia continue se utiliza para saltar el resto del código dentro de un bucle sólo para la iteración actual. El bucle no termina, sino que continúa con la siguiente iteración.

Sintaxis de continue:

```
continue
```