TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE

- Entendiendo las instrucciones condicionales
- ¿Por qué usamos condicionales?
- Siguiendo el flujo de un programa
- Entendiendo un diagrama de flujo
- Creando Subcondiciones
- Evaluación de una expresión
- Creando un programa con condiciones
- Manejando múltiples condiciones con elif
- Manejando condiciones anidadas
- Manejando condiciones de salida
- Utilizando convenciones de nombres en variables (Snake Case)

ENTENDIENDO LAS INSTRUCCIONES CONDICIONALES

En el módulo anterior ya estudiamos las sentencias condicionales, por lo que en este profundizaremos un poco más sobre su uso, flujo y programación.

Como ya sabemos, las estructuras condicionales nos ayudan con la toma de decisiones con respecto a qué hacer si una expresión es verdadera o falsa. Pero acá nos preguntamos:

¿POR QUÉ USAMOS CONDICIONALES?

Cuando creamos un programa, muchas veces no deseamos que se ejecuten todas las instrucciones, pues necesitamos que éstas corran solo si se cumplen las condiciones adecuadas. Haciendo uso de las instrucciones condicionales creamos el flujo del programa, filtrando las instrucciones que no deseamos si las condiciones no son las ideales.

De esta manera podemos crear sistemas complejos que realicen múltiples tareas, según su estructura de toma de decisiones. Esta estructura podemos ilustrarla con un diagrama de flujo, y entendiendo el diagrama podemos seguir **el flujo del programa**.

Para la construcción de las expresiones haremos uso de los operadores lógicos, así como de los operadores de comparación. De esta manera podemos tener una lista de condiciones que se deben cumplir para poder que ocurra una acción.

Anteriormente vimos que la sintaxis del **if** es la siguiente:

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

Y fue ilustrada con el siguiente ejemplo:

```
# Si el número es positivo, imprimimos un mensaje apropiado

num = 3
if num > 0
    print(num, "es un número positivo.")
```

Pero es posible que necesitemos evaluar más de una condición para poder tomar la decisión. Cada una de las condiciones que se deben cumplir las llamaremos **subcondiciones**. Pudiendo tener N cantidad de subcondiciones a evaluar, donde todas y cada una se deben cumplir para ejecutar la acción.

Las subcondiciones pueden ser evaluadas utilizando el operador ***and***, así como con el operador ***or***.

La sintaxis en caso del operador **and** es la siguiente:

```
If subcondicion1 and subcondicion2 and subcondicion3 …
```

Utilizamos el **and** si necesitamos que se cumpla más de una condición.

En el caso de que queramos que se cumpla alguna de las condiciones (cualquiera) para ejecutar la acción debemos utilizar el operador **or**. Para ello la sintaxis es:

```
If subcondicion1 or subcondicion2 or subcondicion3 …
```

Cuando Python realiza la **evaluación de una expresión,** las subcondiciones se evalúan según el orden en que aparecen en el código, así que primero se evalúa la primera subcondición, y en caso de ser verdadera (*true*) se evalúa la condición siguiente, y así sucesivamente. En caso de ser falsa una subcondición, automáticamente se considera falsa toda la expresión del ***if*** o del ***elif***, no haciendo falta evaluar las subcondiciones siguientes. Y en caso de que todas las subcondiciones sean verdaderas, entonces la expresión se considera verdadera.

MANEJANDO MÚLTIPLES CONDICIONES CON ELIF

Muchas veces es necesario evaluar expresiones en cascada, es decir, en caso de que la primera expresión sea falsa, se evaluará una nueva expresión y así sucesivamente. En otros lenguajes de programación tenemos la sentencia **switch**, el creador de Python consideró que **elif** podía cumplir con esa función de forma más sencilla.

Anteriormente habíamos visto que la sintaxis es la siguiente:

```
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
```

En la estructura podemos tener tantos **elif** como sea necesario, los cuales son chequeados según el orden en que aparecen en la estructura.

```
x = 1

if x > 10:
    print(" x es mayor que 10!")
elif x < 10:
     print("x es menor que 10!")
elif x < 20 :
     print("x es menor que 20!")
else:
    print("x es igual a 10")
```

Este ejemplo nos da como resultado:

```
x es menor que 10!
```

Podemos tener una sentencia **if**...**elif**...**else** dentro de otra sentencia **if**...**elif**...**else.** Esto se llama **anidación** en la programación informática.

Cualquier número de estas sentencias puede estar anidado dentro de otro. La indentación es la única manera de averiguar el nivel de anidamiento. Pueden resultar confusas, por lo que deben evitarse a menos que sea necesario.

Veamos un ejemplo de esto. Para aquello evaluemos el siguiente enunciado: ‘’En este programa, introducimos un número, comprobamos si el número es positivo o negativo o cero, y mostramos un mensaje apropiado. Esta vez utilizamos una sentencia if anidada’’

```
num = 4
if num >= 0
    if num == 0:
        print("Cero")
    else:
        print("Número positivo")
else:
    print("Número negativo")
```

Obtenemos como resultado:

```
Número positivo
```

UTILIZANDO CONVENCIONES DE NOMBRES EN VARIABLES

Aun cuando no es obligatorio el uso de ningún sistema en específico para nombrar nuestras variables, funciones, entre otras. es buena práctica seguir algunas convenciones, pues así mejoramos la legibilidad de nuestro código, y facilitamos a otros entender lo que hicimos, e incluso a nosotros mismos cuando queramos hacer revisiones en el futuro.

Existen varios sistemas que se utilizan, dependiendo de que queramos nombrar:

- **lowerCamelCase**: este sistema consiste en poner todas las palabras del nombre sin espacios, escritas en minúscula, excepto la primera letra de cada palabra a partir de la segunda palabra.
- **UpperCamelCase**: este sistema es idéntico al anterior, excepto porque la primera letra de la primera palabra también va en mayúscula.
- **snake_case**: en este caso todas las palabras se escriben en minúsculas, y los espacios se sustituyen por guiones de piso.
- **SCREAMING_SNAKE_CASE**: este caso es igual al anterior, pero con todas las letras en mayúsculas.

Cuando deseamos ponerle nombre a una variable en Python, usualmente el sistema a usar es el **snake_case**.

Si deseamos utilizar simplemente una letra para nombrar a una variable debemos evitar usar la O mayúscula para evitar se confunda con el cero. Evitar usar la l minúscula (ele), pues podría confundirse con un uno, lo que también podría ocurrir con la I mayúscula (i).