

«Talento Tech»

Data Analytics con Python

Clase 02





Clase N° 2 | Conceptos básicos

Temario:

- Condicionales en Programación: Uso y Estructura de if, elif y else
- Manipulación de Diccionarios, listas y tuplas.
- Métodos para manipular estructuras de datos.

Condicionales

Una estructura condicional permite decidir por cuál alternativa seguirá el flujo del programa dependiendo del resultado de la evaluación de una condición. Para establecer condiciones complejas se utilizan los operadores relacionales y lógicos.

Los condicionales nos permiten tomar decisiones en función de eventos, guiando el flujo del programa y permitiendo ejecutar bloques de código específicos si se cumplen ciertas condiciones y, opcionalmente, otros bloques si no se cumplen. Estas estructuras de control son esenciales para crear lógica y adaptabilidad en los programas.

Estructura if

La estructura condicional "if" nos permite ejecutar un bloque de código específico si una determinada condición se cumple, dirigiendo la ejecución del programa según ciertas circunstancias.

Veamos algunos ejemplos:

Si es de noche ➡ puede encender una luz

Si tengo hambre ➡ puedo preparar un almuerzo

Si llueve ➡ uso un paraguas

En estos ejemplos, podemos encontrar bien claro cuál es nuestra condición, de la cual va a depender si se ejecuta nuestra acción o no.

Las condiciones serían las siguientes:

Que sea de noche

Tener hambre

Y que esté lloviendo

Si estas condiciones son verdaderas o falsas, nosotros haremos una cosa u otra.

Tranquilamente, podemos llevar estas decisiones a nuestro lenguaje.



En python, como en tantos otros lenguajes, vamos a utilizar un if, veamos como quedaría nuestra sintaxis:

Código:

```

##
x = 9
y = 5

if(x>y):
    print(f"{x} es mas grande que {y}")
##
Consola:
##
9 es más grande que 5
##
    
```



9 es mas grande que 5

X e Y son variables que usamos con nuestro operador “mayor a” como condición.

Estructura else

Como mencionamos anteriormente, la declaración if en Python se utiliza para operaciones de toma de decisiones. Su estructura permite que se ejecute sólo cuando la condición dada en la declaración if es verdadera.

¿Pero qué pasa si la condición es falsa?

En estos casos existe una palabra reservada traducida como “sino” que se ejecuta cuando la condición del if es falsa. Es decir:

si (la condición se cumple)

pasa algo,

y **sino**

pasa otra cosa.

Esta es la declaración else: Esta declaración es opcional y contiene un código para la condición else.

Cuando querés justificar una condición que resultó falsa, entonces usa la declaración `if else` de Python. Es una de las mas importantes, por lo que te recomiendo que si te queda alguna duda sobre el tema, lo consultes con tus compañeros por el canal de Discord o con tu mentor.

Diagrama de flujo `if... else`



Un ejemplo:

Código:

##

```

a = 4
b = 5

if(a<b):
    print(f"{a} es mas chico que {b}")
else:
    print(f"{a} es mas grande que {b}")
  
```


##

Consola:

##

4 es mas chico que 5

##

 4 es mas chico que 5

- En la línea 1 y 2 declaramos las variables a y b como 4 y 5 respectivamente
- En la línea 3 “preguntamos” si a es menor a b
- Si esta condición es verdadera, entonces imprimimos por consola el texto del print de la línea 4.
- En la línea 5 agrupamos todos los valores que no son verdaderos para la pregunta de la línea 3 y devolverá en este caso, el texto del print de la línea 6.

Cuando “otra condición” no funciona

Puede haber muchos casos en los que tu "condición else" no te dé el resultado deseado. Esto puede hacer que se imprima el resultado incorrecto ya que hay un error en la lógica del programa. En la mayoría de los casos, este error sucede cuando se tiene que justificar más de dos condiciones o declaraciones en un mismo programa.

Código:

##

```

c = 4
d = 4
  
```

```

if(c<d):
    texto = "d es mayor que c"
else:
    texto = "c es mayor que d"
print(texto)
    
```

##

Consola:

##

c es mayor que d

##



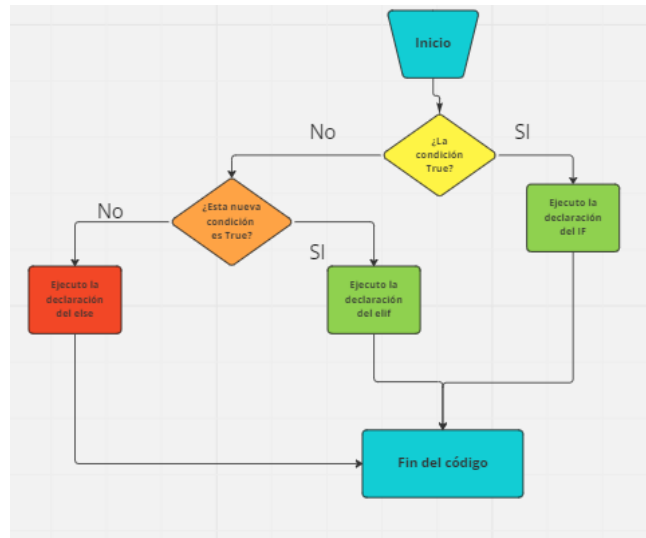
c es mayor que d

Aquí ambas variables valen lo mismo y la salida del programa es "c es mayor que d", lo cual es INCORRECTO . Esto se debe a que verifica la primera condición (condición if), y si falla, imprime la segunda condición (condición else) como predeterminada. Para eso existe la palabra reservada elif, la cuál usaremos después de un if y podemos traducirla como "sino si".

Estructura elif

"Sino si" en Python

Para corregir el error previamente cometido al usar la declaración "else", podemos emplear la instrucción "elif". Al utilizar la estructura condicional "elif", le estamos indicando al programa que ejecute un bloque de código asociado a una segunda o tercera condición o posibilidad cuando la condición anterior resulta falsa o incorrecta



Un ejemplo:

Código:

##

```
c,d = 4, 4
```

```

if(c < d):
    texto = "d es mayor que c"
elif(c == d):
    texto = "d es igual a c"
else:
    texto = "c es mayor que d"

print(texto)
    
```

##

Consola:

##

```
d es igual a c
```

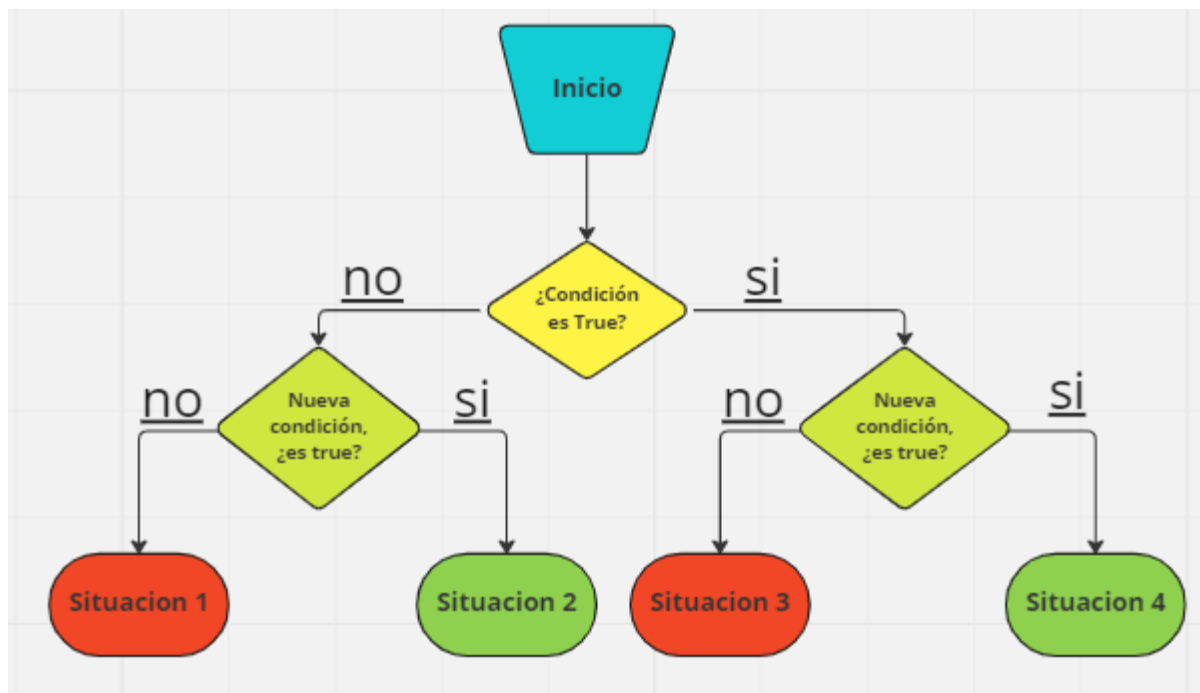
##



⇒ d es igual a c

Anidar declaraciones condicionales

Muchas veces tenemos la necesidad de crear un árbol de preguntas y respuestas para abarcar una mayor cantidad de situaciones y hacer que nuestro código sea más abarcativo. Para esos casos podemos anidar condicionales de la siguiente forma:



Veamos un ejemplo en código como se vería:

Código:

##

```

llueve = True
muchoViento = True
mensaje1 = "Llevo paraguas"
mensaje2 = "Llevo una campera rompevientos"

if (llueve):
    print(mensaje1)
    if(muchoViento):
        print(mensaje1 + " y " + mensaje2)
if(not llueve):
    print("Salgo sin paraguas")
if(not muchoViento):

```



```
print(mensaje2 + " por las dudas")
```

Este ejercicio tiene cuatro variables:

llueve y muchoViento que son True, y dos mensajes de salida de tipo String.

Como está en el diagrama de arriba, esta situación tiene 4 finalidades, cuatro salidas por la terminal distintas que dependen de si las variables “llueve” y “muchoViento” son True o False.

Te invitamos a que copies el código y pruebes cambiando el valor de las variables booleanas para imprimir por la terminal los distintos mensajes.

Detalle

Estamos acostumbrados a que con otros lenguajes podemos utilizar un switch para resolver estos casos. Pero esto no es posible con Python, por lo que tendremos que buscar otras opciones..

Si prestamos atención al código, podemos ver que el condicional llueve, que es True, de la línea 7, es el opuesto al condicional “not llueve” de la línea 11.

Esto como vimos, es lo mismo que decir if y else, así que te propongo en tu código, cambiar la estructura del código para trabajarlo de esta forma. ¡Incluso vas a notar que el diagrama para trabajar con condicionales anidados sería el mismo!

Resumen:

Una estructura condicional en Python es manejada por declaraciones if y vimos otras formas en las que podemos usar declaraciones condicionales en Python cómo else y elif.

if: Se utiliza para imprimir un resultado dependiendo de una sola condición, si esta es verdadera o falsa.

else: Imprime la declaración cuando las condiciones no se cumplen.

elif: Empleado para una tercera posibilidad. Permite verificar múltiples condiciones elif para diferentes posibilidades.



Listas (list).

Las listas en Python son una estructura que nos permite guardar distintos datos en una sola variable. Las listas se definen entre corchetes [], y sus elementos van separados por comas.

Ejemplos de listas:

Código:

##

```
nombres = ["Juan", "Lucia", "Sabrina", "Agustina", "Gabriel"]
edades = [27, 18, 32, 45, 28]
temperaturas = [14.5, 28, 30.5, 19.9]
```

##

Como verás las listas admiten que sus elementos sean distintos tipos de datos, incluso podemos crear listas con elementos de distintos tipos:

Código:

##

```
lista_datos = ["Esteban", 17, True]
```

##

En las listas, cada elemento se ubica en una posición determinada (o índice). Las posiciones comienzan a numerarse desde el número 0. En el ejemplo de la lista nombres tenemos 5 elementos, pero sus posiciones irán del 0 al 4.



Posición	Elemento
0	"Juan"
1	"Lucía"
2	"Sabrina"
3	"Agustina"
4	"Gabriel"

Si de la lista necesitamos un elemento en específico, basta con poner el nombre de la lista y entre corchetes la posición que necesitamos.

Si queremos el elemento "Sabrina", deberíamos poner:

Código:

```
##
```

```
nombres[2]
```

```
##
```

Si luego al elemento "Sabrina" quiero guardarlo en una variable llamada estudiante, deberíamos poner:

Código:

```
##
```

```
estudiante = nombres[2]
```

```
##
```

Además, al utilizar una lista podríamos sobrescribir elementos, es decir reemplazar un elemento por otro.

Queremos reemplazar el elemento de la lista "Agustina" por "Josefina". Lo que deberíamos hacer es, en esa posición de la lista donde se ubica el elemento "Agustina", guardar el

elemento “Josefina”. Según las posiciones, “Agustina” se encuentra en la posición 3, así que en la posición 3 reemplazamos “Agustina” por “Josefina”:

Código:

```
##  
nombres[3] = “Josefina”  
##
```

Tuplas.

¿Qué son las Tuplas?

Las tuplas son una estructura de datos en Python que se utilizan para almacenar una colección ordenada de elementos. Son similares a las listas, pero a diferencia de estas, las tuplas son inmutables, lo que significa que una vez que se han creado, sus elementos no pueden ser modificados, añadidos o eliminados. Las tuplas se definen utilizando paréntesis ().

Sintaxis:

```
##  
datos = ("Carlos", 18, True)  
##
```

Diferencias entre Tuplas y Listas

Mutabilidad:

Tuplas: Inmutables. No se pueden cambiar los elementos después de su creación.

Listas: Mutables. Los elementos pueden modificarse, añadirse o eliminarse.

Sintaxis:

Tuplas: Se definen con paréntesis ().

Listas: Se definen con corchetes [].

Uso y Aplicaciones:

Tuplas: Se utilizan cuando se necesita una secuencia de datos que no deben cambiar durante la ejecución del programa. Por ejemplo, para almacenar coordenadas geográficas, fechas, o como claves en diccionarios.

Listas: Son adecuadas para colecciones de datos que pueden necesitar ser modificadas. Por ejemplo, para almacenar una lista de tareas, una serie de números que deben ser procesados, o cualquier conjunto de datos donde se requiera flexibilidad.

Diccionarios (dict)

Los diccionarios se definen entre llaves {} y sus llaves deben ir entre comillas. El par se pondrá con la estructura 'llave': valor. Y cada par llave: valor, irá separado por comas. Las llaves son cadenas de caracteres, podemos usar tanto comillas dobles "" como comillas simples ' '.

Por ejemplo, podemos usar un diccionario para guardar distintos datos de una persona:

Código:

##

```
datos_messi = {'nombre': "Lionel", 'apellido': "Messi", 'nacionalidad':  
"argentino", 'nombre': "Lionel", 'altura': 1.7, 'fecha nacimiento':  
"24/07/1987"}
```

##

Las llaves de nuestro diccionario son: 'nombre', 'apellido', 'nacionalidad', 'altura' y 'fecha de nacimiento'. Y sus valores son: "Lionel", "Messi", "argentino", 1.7 y "24/07/1987".

Accediendo a sus datos

Ya sabés cómo crear un diccionario. ¿Cómo hacemos para acceder a sus datos?

En el caso de los diccionarios ya no podremos usar posiciones, en lugar de las posiciones usaremos la misma clave para extraer el dato que necesitamos.

Si queremos el valor de la llave 'nombre', indicamos primero el diccionario y luego entre corchetes la llave:



¿Qué sucede cuando hacemos `datos_messi.keys()`?

¿Y cuándo usamos `datos_messi.values()`?

`keys()` y `values()` son dos métodos que podemos aplicar sobre diccionarios.

Si ejecutaste el código anterior te habrás dado cuenta que usar `keys()` sobre el diccionario nos devolverá las llaves del mismo. Y al usar `values()` nos dará sus valores.

Rangos (range).

Python nos permite definir rangos de números. Tenemos tres formas de definir un rango:

Indicando el valor donde finaliza el rango (éste número no estará incluído): el rango comenzará en 0 y terminará un número antes del valor que indicamos, saltando los valores de uno en uno. La forma de declararlo es:

```
range(fin)
```

Toma los valores desde el 0 hasta el fin -1

Ejemplos:

`range(10)` toma los valores de : 0,1,2,3,4,5,6,7,8,9

`range(21)` toma los valores de : 0,1,2,3... hasta el 20

Indicando el inicio y el fin (no incluído): Comenzará en el valor que indiquemos, saltará los valores de uno en uno y finalizará un número antes del valor indicado. La forma de declararlo es:



`range(inicio, fin)`

Toma los valores desde inicio hasta fin -1.

Ejemplos:

`range(2, 10)`: toma los valores 2, 3, 4, 5, 6, 7, 8, y 9.

`range(10, 21)`: toma los valores 10, 11, 12, 13, ..., hasta 20.

Indicando el inicio, el fin y el salto: En los casos anteriores, los valores saltan de uno en uno.

Si agregamos un valor más, éste indica cuando debe saltar al siguiente valor. El fin al igual que en los casos anteriores no está incluido. La forma de declararlo es:

`range(inicio, fin, salto)`

Inicia en el valor inicio, termina en fin-1 y avanza el valor salto.

Ejemplos:

`Range(2, 10, 2)`: toma los valores 2, 4, 6 y 8.

`range(10, 21, 3)`: toma los valores 10, 13, 16 y 19



Desafío N° 2:

Crear un programa que solicite la edad del usuario y muestre por consola si el usuario es mayor de edad o no.



Buenos Aires
aprende
Agencia de Actividades para el Futuro

BA Buenos
Aires
Ciudad