«Talento Tech»

Inteligencia Artificial

Clase 03

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 3 | Estructuras de Datos de Python

Temario:

- Listas, tuplas y diccionarios.
- Estructuras de control: bucles(for - while)
- Métodos para acceder a elementos con listas y diccionarios.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Listas (list)

Las listas se usan para almacenar elementos **ordenados** que pueden cambiar con el tiempo **(mutables**). Son útiles cuando necesitas una secuencia de elementos que puedas agregar, eliminar o modificar sus valores. Pueden almacenar cualquier tipo de dato, incluyendo otras listas. Estos elementos también son **iterables**, lo que significa que podés recorrer sus elementos uno por uno usando una estructura de control, lo que nos permite trabajar fácilmente con los elementos en operaciones repetitivas.
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:

```
nombres = ["Juan", "Lucía", "Mateo", "Martina"]

edades = [17, 15, 14, 19]

datos = ["María", 22, 1.8, ["Verde", "Azul","Amarillo"]]
```

Se dice que una lista es un tipo de dato ordenado porque 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 4 elementos, pero sus posiciones irán del 0 al 3.

Posición	Elemento
0	“Juan”
1	“Lucia”

2	“Mateo”
3	“Martina”

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 **“Mateo”**, deberíamos poner:

```
nombres[2]
```

Si luego al elemento “Mateo” quiero guardarlo en una variable llamada estudiante, deberíamos poner:

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

Si necesito almacenar mas de un elemento en la variable estudiante, puedo hacer un slicing, que me permite almacenar varios elementos de una lista. Si queremos a “Lucia” y a “Mateo” en una lista, los guardo en la variable estudiantes, de la siguiente forma:

```
estudiantes = nombres[1:3]
```

Además, como son mutables, al utilizar una lista podríamos sobreescribir elementos, es decir reemplazar un elemento por otro.

Queremos reemplazar el elemento de la lista “Martina” por “Josefina”. Lo que deberíamos hacer es, en esa posición de la lista donde se ubica el elemento **“Martina”**, guardar el elemento **“Josefina”**. Según las posiciones, “Martina” se encuentra en la posición 3, así que en la posición 3 reemplazamos “Martina” por “Josefina”:

```
nombres[3] = "Josefina"
```

Otros métodos de las tuplas

append

El método append() se usa para agregar un nuevo elemento al final de la lista. Este método es muy útil cuando necesitamos construir una lista de manera dinámica.

```
nombres = ["Juan", "Lucía", "Mateo", "Josefina"]
nombres.append("Sofia")
print(nombres)
```

Cuando ejecutamos la función, la lista *nombres*, tendrá los siguientes elementos:

```
["Juan", "Lucía", "Mateo", "Josefina", "Sofia"]
```

Nota: El método append() modifica la lista original, no es que creamos una nueva variable.

Sort

Ordena los elementos en orden ascendente, pero también podemos ordenarlos en orden descentend utilizando el parámetro “reverse=True”

```
nombres = ["Juan", "Lucía", "Mateo", "Josefina"]
nombres.sort()
print(nombres)
```

Cuando ejecutamos la función, esperamos ver la siguiente salida:

```
["Josefina", "Juan", "Lucía", "Mateo"]
```

y si queremos ordenarlos de forma descendente podemos usar:

```
print(nombres.sort(reverse=True))
```

La salida sería la siguiente:

```
["Mateo", "Lucía", "Juan", "Josefina"]
```

Nota: El método sort() modifica la lista original, no es que creamos una nueva variable. Además solo va a funcionar con listas cuyos elementos sean comparables entre sí, por ejemplo, que todos los elementos de la lista sean del mismo tipo.

Del

Elimina un elemento de la lista en una posición específica. La palabra del es una palabra clave de python y no es método de las listas.

Por ejemplo en nuestro código:

```
nombres = ["Juan", "Lucía", "Mateo", "Josefina"]

del nombres[1]
```

El resultado sería el siguiente:

```
["Juan", "Mateo", "Josefina"]
```

Tuplas (tuple)

Una tupla es una colección **ordenada** de elementos que puede almacenar cualquier tipo de dato, incluyendo otras tuplas. Las tuplas son **inmutables**, lo que significa que no puedes cambiar sus elementos después de haberlas creado. Este tipo de dato es también **iterable**, lo que significa que puedes recorrer sus elementos uno por uno usando un bucle.

A diferencia de las listas, para definir una **tupla** usamos paréntesis ().

Vamos a definir la tupla nombres como:

```
nombres = ("Juan", "Lucía", "Mateo", "Martina")
```

Al igual que en las listas, puedo acceder a los elementos de la tupla indicando la posición del elemento que necesito. Para el elemento “Mateo” que se encuentra en la posición 2, pondríamos:

```
nombres[2]
```

Diferencia con las listas.

Como vimos en las listas podemos sobreescribir un elemento y reemplazarlo por otro. Esto no es posible en las tuplas. Los elementos de las tuplas no se puede modificar una vez creada la tupla.

Lo que sí podremos hacer es reemplazar toda la tupla por otra.

Por ejemplo, si necesitamos que en lugar de “Martina” el elemento sea “Josefina”, tendremos que sobreescribir la tupla:

```
nombres = ("Juan", "Lucía", "Mateo", "Josefina")
```

Diccionarios (dict)

Un diccionario es una estructura de datos en la cual podemos almacenar distintos valores de a pares: una clave y un valor.

Los diccionarios se definen entre **llaves {}** y sus claves deben ir entre comillas. El par se pondrá con la estructura ‘clave’: valor. Y cada par **clave: 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:

```
datos_messi = {'nombre': "Lionel", 'apellido': "Messi", 'nacionalidad': "argentino", 'altura': 1.7, 'fecha_de_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”.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

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

Los diccionarios suelen ser estructuras que vienen con muchísima información, por lo que conocer todas las claves del diccionario, nos resultaría ideal para saber que voy a hacer con esa información. Para eso podés ver todas las claves del diccionario de la siguiente forma:

```
datos_messi.keys()
```

si quiero conocer todas los valores, puedo usar:

```
datos_messi.values()
```

En el caso de los diccionarios es importante conocer las claves, porque ya no podremos usar posiciones para acceder a un elemento. 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 clave:

```
datos_messi['nombre']
```

Rangos (range)

Python nos permite definir rangos de números. Si bien range no es un tipo de dato, es una estructura que se usa muchísimo y nos genera un elemento de tipo range. Tenemos tres formas de definir un rango:

1. Indicando el valor donde finaliza el rango (este número no estará incluido): 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 fin-1. Ejemplos:

- **range(10):** toma los valores del 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9
- **range(21):** toma los valores 0, 1, 2, …, hasta el 20

2. Indicando el inicio y el fin (no incluido): 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.
3. Indicando el inicio, el fin y el salto. En los casos anteriores, los valores saltan de uno en uno. Si agregamos un valor más, esté 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

Estructuras de control : Iteraciones

Bucle for y bucle While

A veces nos sucede, en programación, que debemos ejecutar muchas veces un mismo código para resolver ciertos problemas. Pero se vuelve tedioso tener que escribir varias veces lo mismo, además de que es una muy mala práctica de los desarrolladores. Para evitar eso, vamos a utilizar los “Bucles”
Los bucles pueden ejecutar un mismo bloque de código varias veces hasta que se cumpla una determinada condición. Su uso es bastante común en la programación.

Para poder recorrer un bucle, vamos a tener que tener una “variable extra” (que puede crearse dentro del mismo bucle), la cual nos va a permitir recorrer todos los elementos, ya que será un elemento por vez.

Pero también, debemos tener en cuenta que NO todos los objetos tienen la particularidad de poder ser iterados. Entre los objetos que se pueden iterar se encuentran:

Objetos iterables:

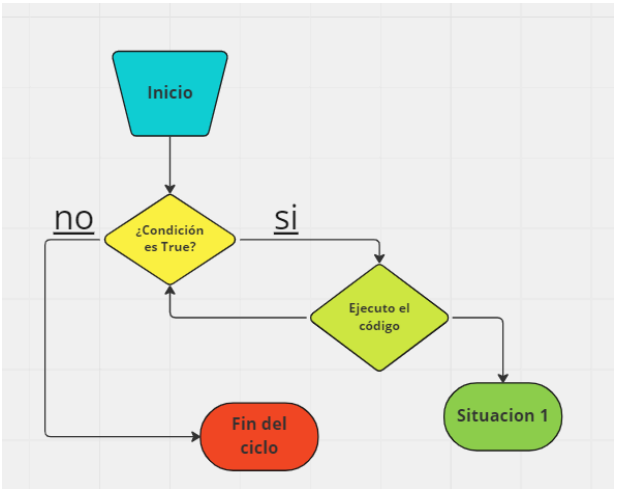
- Cadena de caracteres (strings)
- Listas (list)
- Tuplas (tuple)
- Diccionarios (dict)
- Rangos (range)

Bucle While

En un bucle while (while = mientras), el bloque de código se va ejecutar varias veces hasta que se cumple una determinada condición. Su modo de ejecutarse, es similar a la declaración "If", pero en lugar de ejecutar el bloque de código una vez, regresa al punto donde comenzó el código y repite todo el proceso nuevamente hasta que deje de cumplirse la condición asignada.

Pero usar esta instrucción necesita de un cuidado especial ¡EVITAR QUE SEAN BUCLES INFINITOS! Hay que tener en cuenta que mientras la condición se cumpla, el código va a ejecutarse. Esto es lo que puede dar lugar a bucles infinitos ¡Y como consecuencia harán que nuestra memoria colapse!

Para evitar eso, solemos usar contadores y avisar con un operador lógico, hasta donde queremos que llegue nuestro bucle.



Sintaxis:

```
while(condicion):
    Declaración
```

Ejemplo:

```
y = 0
while (y <= 5):
    print(y)
    y+=1
# Respuesta de consola:
# 0
# 1
# 2
# 3
# 4
# 5
```

En este ejemplo, comenzamos asignando a la variable y el valor de 0. Luego creamos el ciclo while, y preguntamos si Y es menor o igual a 5, y en el caso de que la condición sea verdadera, imprimo el valor de “y”, para luego, sumarle uno.
Entonces en un primer paso, donde Y vale cero, imprimimos el cero y luego le sumamos uno, haciendo que esa variable ahora valga uno.

También podremos hacer, si es posible, un ciclo while de forma más limpia, generando un código de una sola línea de la siguiente forma:

```
x = 5
while x > 0: x-=1; print(x)
# Respuesta de consola:
# 4
# 3
# 2
# 1
# 0
```

En este caso, arrancamos con la variable **“x= 5”** y vamos disminuyendo su valor hasta que x sea mayor a cero.

Bucle For

El bucle for se usa para iterar, realizar varias veces una acción sobre elementos que tengan una secuencia. Estos elementos serán iterables mencionados anteriormente. Se usa cuando se tiene un fragmento de código que se quiera repetir una cierta cantidad de n° veces, por lo que el número de iteraciones de un for está definido de antemano.

La sintaxis tiene como elementos a un objeto que será iterado y a una variable donde iremos guardando los elementos de la secuencia…

```
for Variable in ObjetoIterado:
    Declaración
```

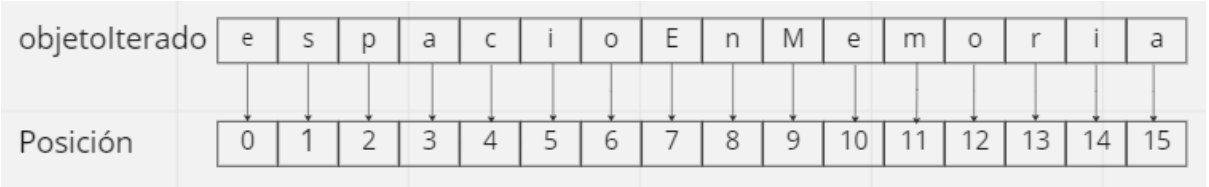
```python
objetoIterado = "espacioEnMemoria"

for variable in objetoIterado:

    print(variable)

print("Fin")
```

En este caso el objeto que iteramos es la variable **"objetoIterado"** que es una variable de tipo String, que tiene el valor de **"espacioEnMemoria"**. Este es un **objeto iterable** el cual tiene una posición para cada uno de sus elementos.



Lo que hacemos es recorrer el objeto iterado e imprimir cada uno de los elementos que guardamos en la variable. Como lo que guardamos en la variable es cada elemento del "objetoIterado", y este está formado por letras que forman una cadena de caracteres, lo que guardamos e imprimimos es letra por letra tantas veces como posiciones tenga el elemento.


```
..  e
    s
    p
    a
    c
    i
    o
    E
    n
    M
    e
    m
    o
    r
    i
    a
    Fin
```

Mientras la posición de la variable no sea mayor, en este caso, a 15, se imprimirá el valor de la variable.

Probá iterar cada uno de los objetos iterables que te comentamos más arriba.

También podemos iterar una acción x cantidad de veces usando la función range.

```python
for x in range(0,10,2):
    print(x)

# Respuesta de consola:
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

```
# 0
# 2
# 4
# 6
# 8
```

En este código estamos diciendo que en “x” guardaremos en memoria los valores que vayan desde 0 hasta 10, yendo de dos en dos. Los valores que tenemos por consola serán entonces, 0,2,4,6,8 sin incluir el 10. Esto se debe a que una vez que llega a diez, el programa se corta y no se llega a ejecutar el código. Una solución a esto es siempre poner un “+1” después del 10 o poner directamente 11.

🐍Enumerate🐍

Vimos que cada objeto iterable tiene una posición para cada uno de sus elementos. ¿Qué pasa si queremos obtener no solo el valor del objeto iterable sino también su posición?¿Para qué nos sirve? . Para esos casos, existe la posibilidad de enumerar cada uno de los elementos.

```
for x, i in enumerate(range(1,21,3)):

    print(f'en la posición {x} está {i}')
```

```
en la posición 0 está 1
en la posición 1 está 4
en la posición 2 está 7
en la posición 3 está 10
en la posición 4 está 13
en la posición 5 está 16
en la posición 6 está 19
```

Esto nos permite más adelante, controlar y guardar los elementos que tengamos en por ejemplo, un diccionario. Para así, buscar los elementos por su posición.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Desafío N° 3:

1. Crea un diccionario vacío.
2. Pide al usuario que ingrese claves y valores (pueden ser, por ejemplo, nombres y edades).
3. Recorre el diccionario con un bucle **`For`** para mostrar las claves y los valores.
4. Pide al usuario que ingrese una clave para buscar en el diccionario y muestra el valor correspondiente (si la clave existe).

   Te doy unas pistas para ayudarte a resolver este desafío:

Pista para el punto 3:

🚀 **¡Explorando el Diccionario!**
Imagina que tu diccionario es un gran libro de recetas, donde cada receta (clave) tiene su lista de ingredientes (valor). Para leer todas las recetas, necesitas recorrer cada página del libro. Utiliza un bucle **for** para hacer esto: **for clave, valor in diccionario.items():**. Esto te permitirá ver cada receta y sus ingredientes en tu pantalla. ¡Es como hacer un recorrido guiado por tu libro de recetas!

Pista para el punto 4:

🔍 **¡Búsqueda en el Diccionario!**
Piensa en tu diccionario como una enciclopedia personal. Si quieres encontrar información específica, necesitas buscar en el índice. Para esto, puedes usar el método **get()** del diccionario. Pide al usuario que ingrese una clave y usa **diccionario.get(clave)** para encontrar la información asociada. ¡Así podrás descubrir lo que estás buscando, siempre y cuando esté en tu enciclopedia!


Buenos Aires
aprende
Buenos
Aires
Ciudad