Introducción a la programación con Python

Listas e iteraciones

Listas

Las listas en Python son un tipo de dato que permite almacenar datos de cualquier tipo. Son mutables y dinámicas.

Las listas en Python son uno de los tipos o estructuras de datos más versátiles del lenguaje, ya que permiten almacenar un conjunto arbitrario de datos. Es decir, podemos guardar en ellas prácticamente lo que sea.

Si vienes de otros lenguajes de programación, se podría decir que son similares a los arrays.

Listas

Existen varias formas de crear una lista

Una lista sea crea con [] separando sus elementos con comas. Una gran ventaja es que pueden almacenar tipos de datos distintos.

```
miLista = [5, "hola", 3, 7]
miLista
[5, 'hola', 3, 7]
```

También se puede crear usando list y pasando un objeto iterable.

```
lista = list("1234")
lista
['1', '2', '3', '4']
```

Propiedades de las listas

Algunas propiedades de las listas:

- Son ordenadas, mantienen el orden en el que han sido definidas
- Pueden ser formadas por tipos arbitrarios
- Pueden ser indexadas con [i].
- Se pueden anidar, es decir, meter una dentro de la otra.
- Son mutables, ya que sus elementos pueden ser modificados.
- Son dinámicas, ya que se pueden añadir o eliminar elementos.

Acceder y modificar listas

Podemos acceder a los elementos de una lista mediante índices.

```
miLista = [5, "hola", 3, 7]
miLista
[5, 'hola', 3, 7]
miLista[1]
'hola'
miLista[0]
5
```

Para modificar los valores podemos asignar otro valor con el operador =

```
miLista[1] = 35
miLista[1]
35
miLista
[5, 35, 3, 7]
```

Métodos de las listas

- append() : añade un elemento al final de la lista.
- extend() : permite añadir una lista a la lista inicial.
- insert() : añade un elemento en una posición o índice determinado.
- remove() : recibe como argumento un objeto y lo borra de la lista.
- pop() : elimina por defecto el último elemento de la lista, pero si se pasa como parámetro un índice permite borrar elementos diferentes al último.
- reverse() : invierte el orden de la lista.
- sort() : ordena los elementos de menor a mayor.
- index() : recibe como parámetro un objeto y devuelve el índice de su primera aparición.

El bucle for

En general, un bucle es una estructura de control que repite un bloque de instrucciones.

Un bucle for es un bucle que repite el bloque de instrucciones un número predeterminado de veces. El bloque de instrucciones que se repite se suele llamar cuerpo del bucle y cada repetición se suele llamar iteración.

La sintaxis de un bucle for es la siguiente:

```
for variable in elemento iterable (lista, cadena, range, etc.):

    cuerpo del bucle
```

No es necesario definir la variable de control antes del bucle, aunque se puede utilizar como variable de control una variable ya definida en el programa.

El bucle for

El cuerpo del bucle se ejecuta tantas veces como elementos tenga el elemento recorrible (elementos de una lista o de un range(), caracteres de una cadena, etc.)

```
print("Comienzo")
for i in [0, 1, 2]:
    print("Hola ", end="")
print()
print("Final")
```

```
Comienzo
Hola Hola Hola
Final
```

El bucle for

Si la lista está vacía, el bucle no se ejecuta ninguna vez. Por ejemplo:

```
print("Comienzo")
for i in []:
    print("Hola ", end="")
print()
print("Final")
```

```
Comienzo

Final
```

El bucle for

En los ejemplos anteriores, la variable de control "i" no se utilizaba en el bloque de instrucciones, pero en muchos casos sí que se utiliza.

Cuando se utiliza, hay que tener en cuenta que la variable de control va tomando los valores del elemento recorrible. Por ejemplo:

```
print("Comienzo")
for i in [3, 4, 5]:
    print(f"Hola. Ahora i vale {i} y su cuadrado {i ** 2}")
print("Final")
```

```
Comienzo
Hola. Ahora i vale 3 y su cuadrado 9
Hola. Ahora i vale 4 y su cuadrado 16
Hola. Ahora i vale 5 y su cuadrado 25
Final
```

El bucle for

Podemos iterar una cadena de texto (string).

```
texto = "Esto es un STRING"
for i in texto:
    print(i)

E
s
t
o

e
s

u
n

S
T
R
I
N
G
```

El bucle for

También se puede iterar sobre una lista.

```
for i in miLista:
    print(i)

4
2
8
100
palabra
35
```

For anidado

Es posible anidar los for, esto quiere decir, ejecutar un for dentro de otro.

Nos puede servir para iterar sobre un elemento iterable que esté formado por otros elementos iterables, como una lista en la que todos sus elementos son también listas.

```
miLista=[[1,2,3],[2,3],[8,4,98,13]]
for i in miLista:
    for j in i:
        print(j)


1
2
3
2
3
8
4
98
13
```

```
for i in miLista:
    print(i)
    for j in i:
        print(j)

[1, 2, 3]
1
2
3
[2, 3]
2
3
[8, 4, 98, 13]
8
4
98
13
```

Range

Algo muy común es iterar un número de veces entre 0 y n. Por ejemplo si tenemos que realizar 5 veces la misma acción:

```
for i in [0,1,2,3,4]:
    print("iteracion")

iteracion
iteracion
iteracion
iteracion
iteracion
```

Podemos hacerlo de esta forma, creando una lista de 5 elementos sobre los que iterar, pero no es lo más recomendable.

Range

El range(n) crea una secuencia de números desde 0 hasta n-1

```
for i in range(5):
    print("iteracion: ", i)

iteracion:  0
iteracion:  1
iteracion:  2
iteracion:  3
iteracion:  4
```

Range

Este método acepta otros argumentos en su llamada

range(inicio, fin, salto)

El inicio es el número en el que empezará el rango.

El fin es el número-1 en el que acabará el rango.

El salto es el valor que varía en cada iteración.

```
for i in range(5,25,2):
    print(i)

5
7
9
11
13
15
17
19
21
23
```

Range

No es obligatorio que el salto siempre sea hacia adelante.

```
for i in range(25,5,-2):
    print(i)

25
23
21
19
17
15
13
11
9
7
```