


Open · Opened 4 days ago by  **Rubén Montero**

Herramientas para usar en listas

Resumen

- Aprenderemos a usar algunas herramientas útiles para trabajar con listas
- Añadiremos una función a `loops.py`

Descripción

Hasta ahora, nuestro manejo de las *listas* se limita a un bucle `for`.

¡Podemos hacer muchas más cosas!

`in`

En la tarea anterior hemos recorrido una lista para *buscar* un elemento. En concreto, el nombre de un **dinosaurio**.

¡No hace falta!

En ejercicios anteriores se mencionó de pasada que el operador `in` sirve para verificar si *un elemento* está en *una lista*:

```
'a' in ['a', 'b', 'c']
```

Probémoslo.

Añadamos una **nueva función** `find_dino2` a `loops.py`:

```
def find_dino2(dinosaur):  
    if dinosaur in ["Triceratops", "Diplodocus", "Pterodáctilo"]:  
        return True  
    else:  
        return False
```

Un *casi* error muy común

Código como este:

```
if condicion == True:  
    return True  
else:  
    return False
```

...es *boilerplate* típico de principiante en cualquier lenguaje de programación, pues puede escribirse más brevemente:

```
return condicion == True
```

Ya verás.

Modifica `find_dino2` y **déjalo** así:

```
def find_dino2(dinosaur):  
    return dinosaur in ["Triceratops", "Diplodocus", "Pterodáctilo"]
```

¡Qué compacto!

`len`

Para hallar la longitud (tamaño) de una *lista* en Python se usa `len(...)`.

Es una abreviatura de `length` (longitud).

Añade una nueva función a `loops.py`:

```
def example_length():
    list1 = ["Alice", "Bob"]
    size1 = len(list1)
    print("El tamaño de la primera lista es: " + str(size1))

    list2 = [None, None, None, None]
    size2 = len(list2)
    print("El tamaño de la segunda lista es: " + str(size2))

    list3 = [[1, 2], [3, 4]]
    size3 = len(list3)
    print("El tamaño de la tercera lista es: " + str(size3))
```

Acceder a un elemento

Para acceder a un elemento de una lista, usamos los corchetes (`[]`) y un *índice*, como en otros lenguajes de programación.

```
>>> numeros = [8, 14, 3]
>>> numeros[0]
8
>>> numeros[2]
3
```

¡Podemos liarla especificando un índice **inválido**!

```
>>> numeros[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Sublistas

Algo **muy interesante** de Python es que podemos acceder a un *fragmento* de una lista con mucha facilidad, empleando dos puntos (`:`):

```
>>> numeros[0:0]
[]
>>> numeros[0:1]
[8]
>>> numeros[0:2]
[8, 14]
>>> numeros[0:3]
[8, 14, 3]
>>> numeros[0:4]
[8, 14, 3]
>>> numeros[1:3]
[14, 3]
```

Como ves, con esta sintaxis no saltan excepciones `IndexError`. Tan sólo se devuelven listas vacías.

¡Incluso se permite un índice **negativo**!

Significará *empezar por el final de la lista*:

```
>>> numeros[-1]
3
>>> numeros[-2]
14
```

Añadamos una función

Con respecto a esta última parte, no profundizaremos mucho, aunque es interesante conocer el potencial.

Basta con que añadas una nueva **función** a `loops.py` que **reciba un parámetro**. Será un *índice*.

La función se llamará `retrieve_value` y:

- Recuperará el valor del elemento de la lista `[4, 8, -35, "Pepe Depura", 45]` especificado por el *índice* del **parámetro**
 - Lo devolverá usando `return`

- En caso de ser un índice inválido, **no** fallará. Devolverá **None**

Por último

Si has ido siguiendo los pasos y lo que se pedía que añadieses a `loops.py`, la tarea está completada.

Verifica que tu código pasa el *test* asociado a la tarea.

Haz `commit` y `push` para subir los cambios al repositorio.



Rubén Montero @ruben.montero changed milestone to [%Sprint 3](#) 4 days ago