Colecciones

Fundamentos de Programación

FIEC04341

Sesión 01

Agenda

- Características de las colecciones
- Tipos de colecciones
- Operaciones con colecciones

Terminología

Terminología

- **Una colección** es un tipo de dato que agrupa varios elementos en una misma unidad. Se utilizan para almacenar, recuperar, manipular y comunicar una agregación de datos.

- Fuente: https://www.iconfinder.com

Terminología

- Representan ítems que forman una agrupación natural como una mano de poker (una colección de cartas), una carpeta de correos (una colección de cartas), un directorio telefónico (un mapeo de nombres hacia números telefónicos).

Tipos de colecciones

Tipos de colecciones

- Python tiene los siguientes tipos de colecciones:
  - Listas
  - Tuplas
  - Diccionarios

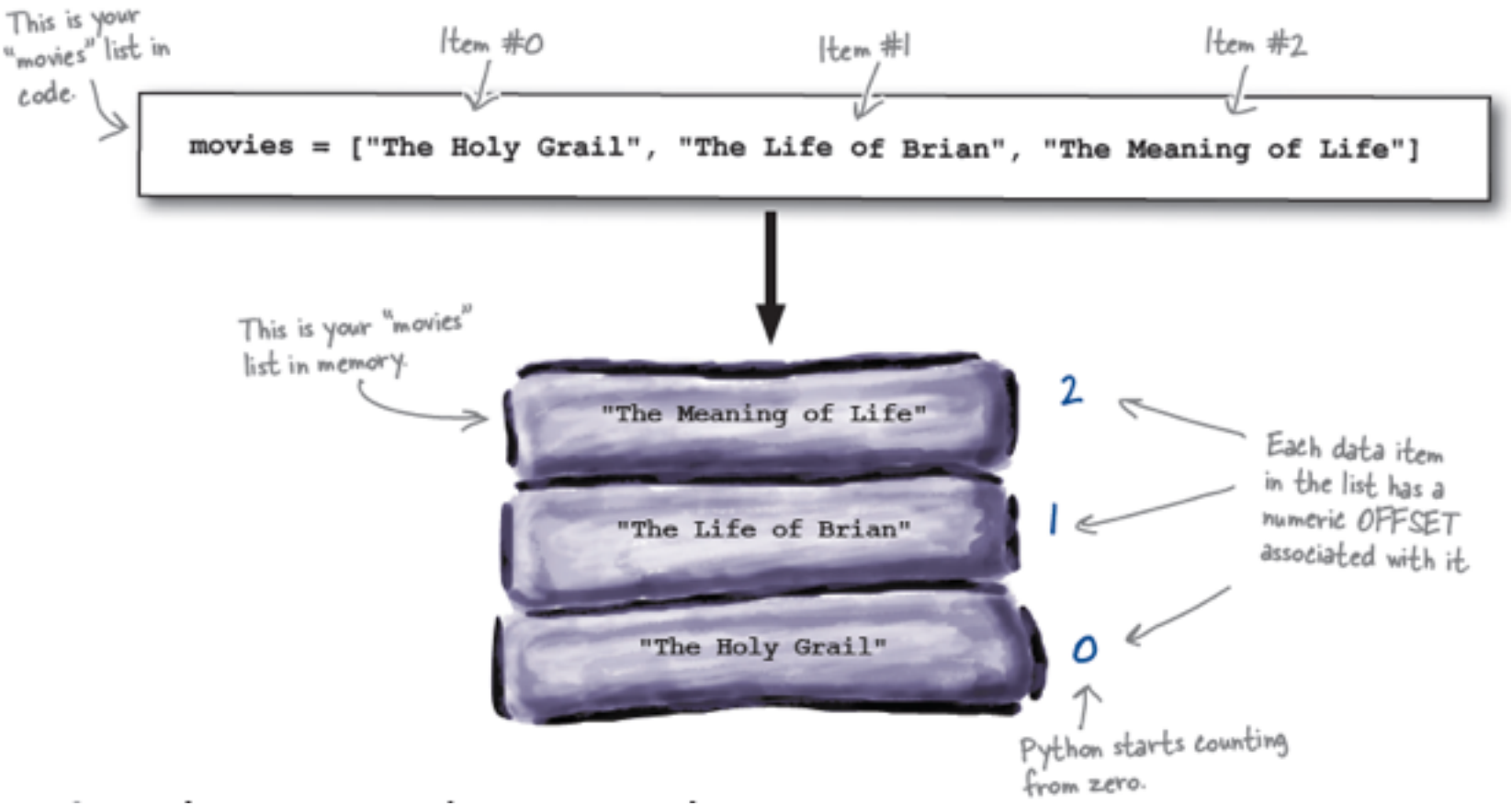
Listas

Listas

- La lista es un tipo de colección ordenada. Pueden contener cualquier tipo de dato: número, cadenas, booleanos, … y también listas.
- Crear una lista es tan sencillo como indicar entre corchetes y separados por comas, los valores que se desea incluir en la lista:

```
movies = ["The Holy Grail",
          "The Life of Brian",
          "The Meaning of Life"]
```

Listas


This is your "movies" list in code.
Item #0
Item #1
Item #2
movies = ["The Holy Grail", "The Life of Brian", "The Meaning of Life"]
This is your "movies" list in memory.
"The Meaning of Life"
"The Life of Brian"
"The Holy Grail"
2
1
0
Each data item in the list has a numeric OFFSET associated with it
Python starts counting from zero.

Listas

- Para acceder a un ítem de una lista se debe colocar el nombre de la lista seguida de su índice entre corchetes.

Listas

- Una misma lista puede contener múltiples tipos de dato.

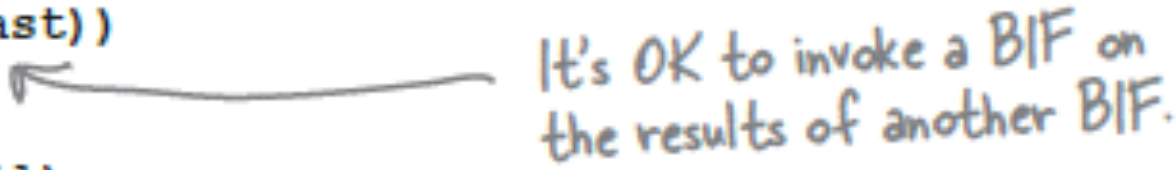
```
["The Holy Grail", 1975, "The Life of Brian", 1979, "The Meaning of Life", 1983]
```

```
l = [22, True, "una lista", [1, 2]]
```

Listas

- Para obtener el número de elementos contenidos en una lista se puede usar la función embebida len()

```
>>> cast = ["Cleese", 'Palin', 'Jones', "Idle"]
>>> print(cast)
['Cleese', 'Palin', 'Jones', 'Idle']
>>> print(len(cast))
4
>>> print(cast[1])
Palin
```

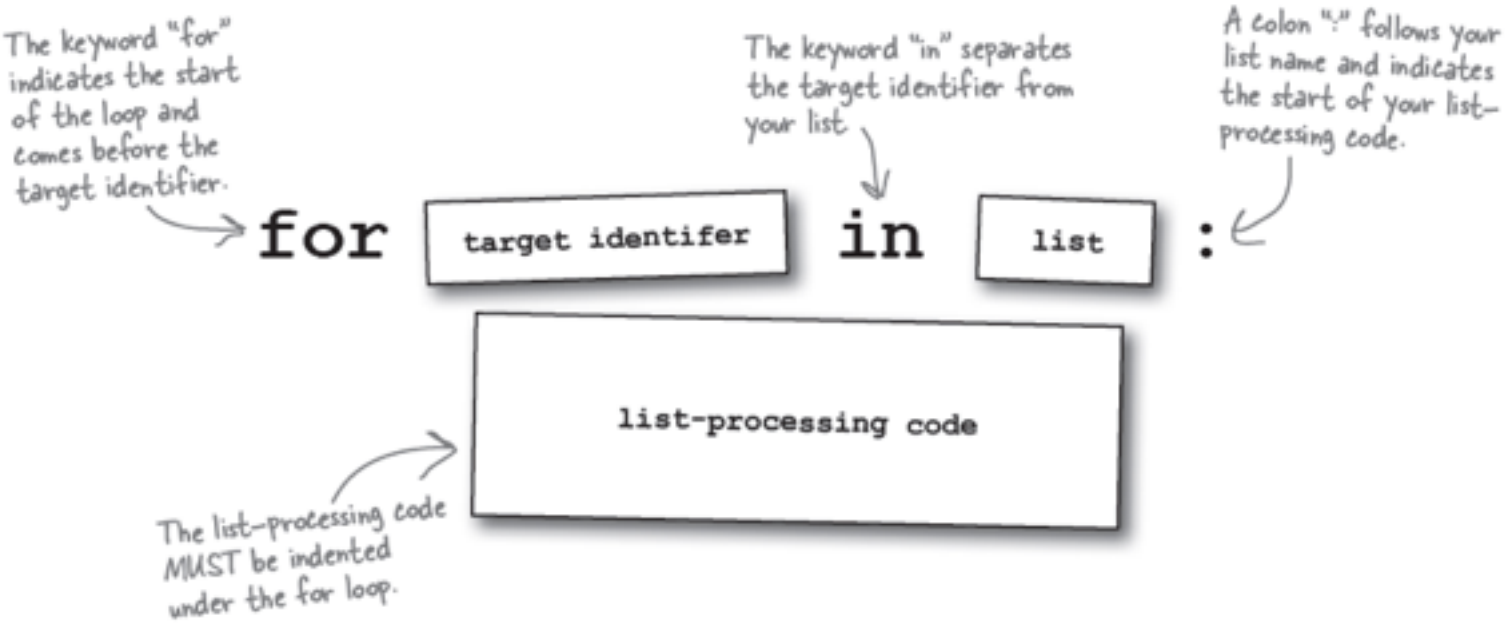


- BIF: Built-in function

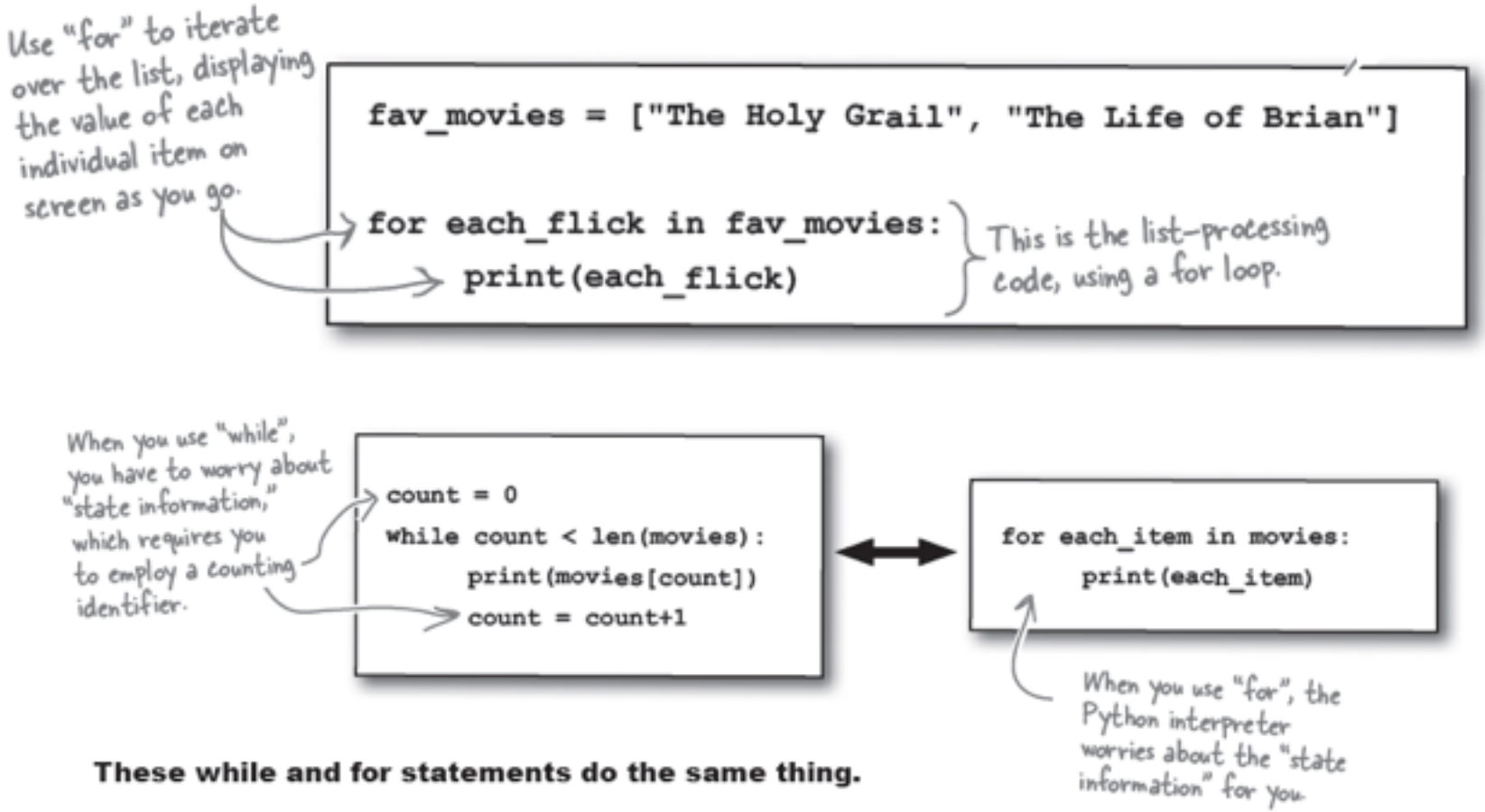
Operaciones con listas

Operaciones con listas

- Para recorrer una lista con lazos se puede utilizar el lazo for.



Operaciones con listas

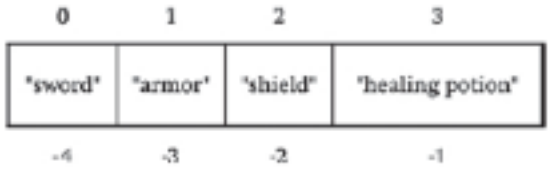


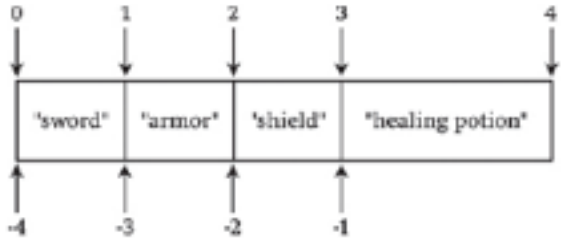
Operaciones con listas

- El operador in sirve para verificar si un elemento es parte de una lista.

```
inventory = ["sword", "armor", "shield", "healing potion"]

# test for membership with in
if "healing potion" in inventory:
    print("You will live to fight another day.")
```





Operaciones con listas

- Para obtener un parte de una lista se debe escribir entre corchetes la ubicación inicial y final del fragmento.

```
inventory = ["sword", "armor", "shield", "healing potion"]

# display a slice
start = int(input("\nEnter the index number to begin a slice: "))
finish = int(input("Enter the index number to end the slice: "))

print("inventory[", start, ":", finish, "] is", end=" ")
print(inventory[start:finish])

input("\nPress the enter key to continue.")
```

Operaciones con listas

- Para concatenar dos listas se puede utilizar el operador +

```
inventory = ["sword", "armor", "shield", "healing potion"]

# concatenate two lists
chest = ["gold", "gems"]
print("You find a chest which contains:")
print(chest)
print("You add the contents of the chest to your inventory.")
inventory += chest
print("Your inventory is now:")
print(inventory)

input("\nPress the enter key to continue.")
```

Operaciones con listas

- Se puede asignar un nuevo valor a un elemento ya existente por medio de su índice.

```
# assign by index
print("You trade your sword for a crossbow.")
inventory[0] = "crossbow"
print("Your inventory is now:")
print(inventory)

input("\nPress the enter key to continue.")
```

- Recuerde que no puede crear un nuevo elemento de esta manera, es decir, asignar un nuevo valor a un elemento no existente.

Operaciones con listas

- Para eliminar un elemento de una lista por medio de del

```
# delete an element
print("In a great battle, your shield is destroyed.")
del inventory[2]
print("Your inventory is now:")
print(inventory)

input("\nPress the enter key to continue.")
```

Métodos de listas

- Las listas tienen métodos que permiten manipularlas. Así podemos agregar, remover un elemento basado en su valor, ordenar una lista, e incluso ponerla en orden reverso.

Métodos de listas

- Si se desea agregar un elemento al final de la lista se puede utilizar el método append().
- Si lo que se requiere es remover el último se utiliza pop()

```
>>> cast.append("Gilliam")
>>> print(cast)
['Cleese', 'Palin', 'Jones', 'Idle', 'Gilliam']
>>> cast.pop()
'Gilliam'
>>> print(cast)
['Cleese', 'Palin', 'Jones', 'Idle']
```

Methods are invoked using the common "." dot notation.

Métodos de listas

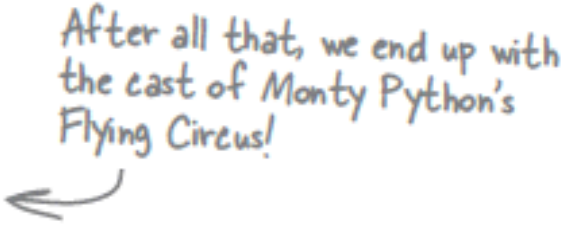
- Si se desea agregar una colección al final de una lista se puede usar el método extend()

```
>>> cast.extend(["Gilliam", "Chapman"])
>>> print(cast)
['Cleese', 'Palin', 'Jones', 'Idle', 'Gilliam', 'Chapman']
```

Métodos de listas

- Por otro lado, si lo que se necesita es remover o agregar un elemento en una posición específica se utiliza las funciones remove() e insert(), respectivamente.

```
>>> cast.remove("Chapman")
>>> print(cast)
['Cleese', 'Palin', 'Jones', 'Idle', 'Gilliam']
>>> cast.insert(0, "Chapman")
>>> print(cast)
['Chapman', 'Cleese', 'Palin', 'Jones', 'Idle', 'Gilliam']
```



Métodos de listas

- El método sort() ordena los elementos de una lista en orden ascendente – el valor más pequeño primero.
- No obstante, si se desea ordenar los valores de la lista en orden descendente, se puede pasar True al parámetro reverse:

```
# sort scores
elif choice == "4":
    scores.sort(reverse=True)
```