Listas en Python

Alumnos

Agustín Salinas
Rubio Fabricio

Introducción

En esta investigación se mostrará y explicarán algunos: métodos de las listas de Python, funciones para iterables (aplicados en listas)

Palabras Clave

Python, Listas, Métodos, Funciones, Útiles, Iterables

Desarrollo

Para poder definir métodos de utilidad para las listas, primero se debería conocer que las listas de Python son objetos iterables. Lo que permite usar la mayoría de funciones para iterables en listas (algunas no, como el *next*)

¿Qué es un objeto iterable?

Son objetos sobre los cuales se puede iterar, esto significa que estos objetos devuelven un valor a la vez, permitiendo así ser usados en un loop *for ... in*.

Detrás de bambalinas un objeto se vuelve iterable cuando posee un método *__iter__* o *__getitem__*

A partir de ahora todas las partes de código mostradas serán parte de un mismo código, solo que se dividirá en fragmentos como ejemplos

```
class Materia:
    __alumnos: int
    __nombre: str

    def __init__(self, nombre: str, alumnos: int):
        self.__alumnos = alumnos
        self.__nombre = nombre

    def get_alumnos(self) -> int:
        return self.__alumnos

    def get_nombre(self) -> str:
        return self.__nombre

    def __repr__(self) -> str:
        return f"{self.__nombre} ({self.__alumnos} alumnos)"
```

Sobrecarga *__gt__* y la función para iterables *max*

La función *max* devuelve el valor más grande en un iterable, sin embargo la función *max* no se podría usar en una lista de instancias de la clase Materia, dado que no hay manera de comparar cual es más grande que otra

La sobrecarga ***__gt__*** (aplicada en la clase Materia) nos permite programar el comportamiento del operador > cuando el valor a la izquierda del operador sea una instancia de la clase Materia

Si se aplicara la sobrecarga ***__gt__*** en la clase materia, tal que:

```
    def __gt__(self, other: Materia) -> bool:
        if(type(other) != Materia):
            raise TypeError("No se puede comparar una materia con un objeto de otro
tipo")

        return self.__alumnos > other.get_alumnos()
```

Esto permitiría usar la función *max* en una lista de materias, ejemplo:

```
lista: list[Materia] = [
    Materia('Algoritmos y resolución de problemas', 120),
    Materia('Programación procedural', 90),
    Materia('Programación orientada a objetos', 70),
]

print('Resultado:')
print(max(lista))

"""
Resultado:
Algoritmos y resolución de problemas (120 alumnos)
"""
```

Método *list.extend(iterable)*

Añade a la lista todos los valores devueltos por el iterable, ejemplo:

```
lista.extend([
    Materia('Estructura de Datos y Algoritmos', 50),
    Materia('Paradigmas de Lenguajes', 45),
    Materia('Algoritmos Numéricos', 40),
])

print('Resultado:')
for materia in lista:
    print(materia)

"""
Resultado:
Algoritmos y resolución de problemas (120 alumnos)
Programación procedural (90 alumnos)
```

```
Programación orientada a objetos (70 alumnos)
Estructura de Datos y Algoritmos (50 alumnos)
Paradigmas de Lenguajes (45 alumnos)
Algoritmos Numéricos (40 alumnos)
"""
```

Función *map(function, iterable, ...)* para iterables

Esta función devuelve un iterable cuyos valores son los que devuelve la función function cuando es ejecutada por cada elemento del iterable, ejemplo:

```
iterableNombres = map(lambda x: x.get_nombre(), lista)

print('Resultado: ')
for elemento in iterableNombres:
    print(elemento)

"""
Resultado:
Algoritmos y resolución de problemas
Programación procedural
Programación orientada a objetos
Estructura de Datos y Algoritmos
Paradigmas de Lenguajes
Algoritmos Numéricos
"""
```

Método reverse

Este método invierte el orden de elementos en la lista, ejemplo:

```
lista.reverse()

print('Resultado: ')
for elemento in lista:
    print(elemento)

"""
Resultado:
Algoritmos Numéricos (40 alumnos)
Paradigmas de Lenguajes (45 alumnos)
Estructura de Datos y Algoritmos (50 alumnos)
Programación orientada a objetos (70 alumnos)
Programación procedural (90 alumnos)
Algoritmos y resolución de problemas (120 alumnos)
"""
```

Conclusiones

Dado que estas fueron apenas unas pocas de las muchas funciones para iterables y métodos de funciones, es simple de ver que en Python hay cientos de formas para interactuar con las listas

Bibliografía

[1] docs.python.org/datastructures
[2] w3schools/python
[3] docs.python.org/functions
[4] pythonlikeyoumeanit.com/iterables