Programación Avanzada
Daniel Gregorio Longino
Tarea 3

```
__add__(self, value, /)
    Return self+value.
```

Regresa una nueva lista que resulta ser la concatenación de las dos listas.

```
>>> [1,2]+[3,4]
[1, 2, 3, 4]
>>> [1,2].__add__([3,4])
[1, 2, 3, 4]
```

```
__contains__(self, key, /)
    Return key in self.
```

Indica si un elemento (*key*) está contenido en una lista. Regresa *True* si lo está y *False* si no los está.

```
>>> 3 in [1,2,3]
True
>>> [1,2,3].__contains__(4)
False
```

```
__delitem__(self, key, /)
    Delete self[key].
```

Elimina el elemento ubicado en el indice descrito.

```
>>> x = [2,4,8,10,12]
>>> del x[3]
>>> x
[2, 4, 8, 12]
```

```
__eq__(self, value, /)
    Return self==value.
```

Indica si dos listas son iguales.

```
>>> [1,2,3]==[1,2,3]
True
>>> [1,2]==[1,2,3,4]
False
```

```
__ge__(self, value, /)
    Return self>=value.
```

Compara dos listas de igual tamaño entrada a entrada, y regresa *True* si la entrada i-ésima de la primera lista es mayor o igual a la entrada i-ésima de la segunda lista para todo i. Y regresa *False* en otro caso.

```
>>> [1,2]>=[1,0]
True
>>> [2,3,4]>=[3,2,1]
False
```

```
__getattribute__(self, name, /)
    Return getattr(self, name).
```

Devuelve el valor del atributo nombrado de un objeto. Si no se encuentra, devuelve el valor predeterminado proporcionado a la función.

```
1  class Person:
2      age = 23
3      name = "Adam"
4
5  person = Person()
6  print('The age is:', getattr(person, "age"))
7  print('The age is:', getattr(person, "name"))
8
```

```
The age is: 23
The age is: Adam
[Finished in 654ms]
```

```
__getitem__(...)
    x.__getitem__(y) <==> x[y]
```

Regresa el elemento de la lista ubicado en el indice descrito.

```
>>> x = [1,2,3,4]
>>> x[2]
3
>>> x.__getitem__(3)
4
```

```
__gt__(self, value, /)
    Return self>value.
```

Compara dos listas de igual tamaño entrada a entrada, y regresa *True* si la entrada i-ésima de la primera lista es mayor a la entrada i-ésima de la segunda lista para todo i. Y regresa *False* en otro caso.

```
>>> [1,2]>[0,1]
True
>>> [1,2]>[1,2]
False
```

```
__iadd__(self, value, /)
    Implement self+=value.
```

Devuelve la concatenación de dos listas. Es necesario importar el módulo operator.

```
>>> import operator
>>> operator.iadd([1,2],[3,4])
[1, 2, 3, 4]
```

```
__imul__(self, value, /)
    Implement self*=value.
```

Concatena *key* veces la misma lista.

```
>>> [1,2].__imul__(3)
[1, 2, 1, 2, 1, 2]
>>> [1,2,3]*4
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
__init__(self, /, *args, **kwargs)
    Initialize self.  See help(type(self)) for accurate signature.
```

Cuando se crea un nuevo objeto de una clase, Python llama automáticamente al método __init__() para inicializar los atributos del objeto.

```
1  class Person:
2      def __init__(self, name, age):
3          self.name = name
4          self.age = age
5
6
7  if __name__ == '__main__':
8      person = Person('John', 25)
9      print(f"I'm {person.name}. I'm {person.age} years old.")
10
```

```
__iter__(self, /)
    Implement iter(self).
```

Crea un objeto de la clase list_iterator, el cual es un iterador. Con el método __next__ de la clase list_iterator, se recorre toda la lista original.

```
>>> x = [1, 2, 3].__iter__()
>>> x.__next__()
1
>>> x.__next__()
2
>>> x.__next__()
3
```

```
__le__(self, value, /)
    Return self<=value.
```

Compara dos listas de igual tamaño entrada a entrada, y regresa *True* si la entrada i-ésima de la primera lista es menor o igual a la entrada i-ésima de la segunda lista para todo i. Y regresa *False* en otro caso.

```
>>> [1,2]<=[2,2]
True
>>> [3,4]<=[2,4]
False
```

```
__len__(self, /)
    Return len(self).
```

Devuelve el número de elementos de la lista.

```
>>> x = [1, 2, 3]
>>> len(x)
3
>>> x.__len__()
3
```

```
__lt__(self, value, /)
    Return self<value.
```

Compara dos listas de igual tamaño entrada a entrada, y regresa *True* si la entrada i-ésima de la primera lista es menor a la entrada i-ésima de la segunda lista para todo i. Y regresa *False* en otro caso.

```
>>> [1,2]<[1,3]
True
>>> [1,2]<[0,2]
False
```

```
__mul__(self, value, /)
    Return self*value.
```

Devuelve la lista concatenada con ella misma tantas veces como se indique.

```
>>> x = [1, 2, 3]
>>> x.__mul__(3)
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> 4*x
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
__ne__(self, value, /)
    Return self!=value.
```

Regresa *True* si las dos listas que se están comparando no son iguales y *False* en otro caso.

```
>>> [1,2]!=[1,3]
True
>>> [1,2]!=[1,2]
False
```

```
__repr__(self, /)
    Return repr(self).
```

Transforma la lista completa en una cadena.

```
>>> x = [1, 2, 3]
>>> x.__repr__()
'[1, 2, 3]'
```

```
__reversed__(self, /)
    Return a reverse iterator over the list.
```

Reordena los elementos de una lista, el que era el primer elemento ahora lo manda como último elemento, el que era segundo elemento, lo manda como el penúltimo elemento, etc.

```
>>> list(reversed([1,2]))
[2, 1]
```

```
__rmul__(self, value, /)
    Return value*self.
```

Al igual que el método __mul__, devuelve la lista concatenada con ella misma el número de veces que se indique.

```
>>> x = [1, 2, 3]
>>> x.__rmul__(4)
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
__setitem__(self, key, value, /)
    Set self[key] to value.
```

Cambia el elemento ubicado en el indice descrito por el nuevo valor que se le está asignando.

```
>>> x = [1,2,3,4,5]
>>> x[0]=6
>>> x
[6, 2, 3, 4, 5]
```

```
__sizeof__(self, /)
    Return the size of the list in memory, in bytes.
```

Devuelve el tamaño de la lista en memoria, la cual está en bytes.

```
>>> x = [1, 2, 3]
>>> x.__sizeof__()
64
```

```
append(self, object, /)
    Append object to the end of the list.
```

Agrega el elemento indicado al final de la lista.

```
>>> x = [1,2,3]
>>> x.append(4)
>>> x
[1, 2, 3, 4]
```

```
clear(self, /)
    Remove all items from list.
```

Elimina los elementos de la lista.

```
>>> x = [1, 2, 3, 4]
>>> x.clear()
>>> x
[]
```

```
copy(self, /)
    Return a shallow copy of the list.
```

Crea una copia de una lista.

```
>>> x = [1,2,3,4]
>>> y = x.copy()
>>> y
[1, 2, 3, 4]
```

```
count(self, value, /)
    Return number of occurrences of value.
```

Retorna el número de ocurrencias de un elemento de la lista.

```
>>> x = [1, 2, 3, 4, 1, 2, 1]
>>> x.count(1)
3
>>> x.count(2)
2
>>> x.count(3)
1
```

```
extend(self, iterable, /)
    Extend list by appending elements from the iterable.
```

Extiende la lista agregando elementos de un iterable.

```
>>> x = [1, 2, 3, 4]
>>> x.extend([5, 6, 7])
>>> x
[1, 2, 3, 4, 5, 6, 7]
```

```
index(self, value, start=0, stop=9223372036854775807, /)
    Return first index of value.

    Raises ValueError if the value is not present.
```

Devuelve el primer índice de cierto elemento. Si el elemento no pertenece a la lista, devuelve un error.

```
>>> x = [3, 5, 4, 2, 1]
>>> x.index(5)
1
>>> x.index(2)
3
```

```
insert(self, index, object, /)
    Insert object before index.
```

Agrega el elemento indicado en el indice descrito.

```
>>> x = [1, 2, 3, 4]
>>> x.insert(3,"a")
>>> x
[1, 2, 3, 'a', 4]
```

```
pop(self, index=-1, /)
    Remove and return item at index (default last).

    Raises IndexError if list is empty or index is out of range.
```

Elimina y retorna el elemento de la lista dado cierto índice. Si la lista está vacía, ocurre un error.

```
>>> x = [3, 5, 4, 2, 1]
>>> x.pop()
1
>>> x
[3, 5, 4, 2]
>>> x.pop(1)
5
>>> x
[3, 4, 2]
```

```
remove(self, value, /)
    Remove first occurrence of value.

    Raises ValueError if the value is not present.
```

Elimina la primera ocurrencia del elemento indicado. Regresa error si el elemento indicado no está en la lista.

```
>>> x = [1, 2, "a", 1]
>>> x.remove(1)
>>> x
[2, 'a', 1]
```

```
reverse(self, /)
    Reverse *IN PLACE*.
```

Devuelve la lista en el orden inverso a la original.

```
>>> x = [3, 5, 4, 2, 1]
>>> x.reverse()
>>> x
[1, 2, 4, 5, 3]
```

```
sort(self, /, *, key=None, reverse=False)
    Stable sort *IN PLACE*.
```

En el caso de una lista donde sus entradas son números reales, los ordena de menor a mayor.

```
>>> x = [3, 5, 4, 2, 1]
>>> x.sort()
>>> x
[1, 2, 3, 4, 5]
```

```
Static methods defined here:

__new__(*args, **kwargs) from builtins.type
    Create and return a new object.  See help(type) for accurate signature.
```

__new__() es un método estático de la clase objeto. Cuando crea un nuevo objeto llamando a la clase, Python llama al método __new__() para crear el objeto primero y luego llama al método __init__() para inicializar los atributos del

objeto.

```python
class Person:
    def __new__(cls, first_name, last_name):
        # create a new object
        obj = super().__new__(cls)

        # initialize attributes
        obj.first_name = first_name
        obj.last_name = last_name

        # inject new attribute
        obj.full_name = f'{first_name} {last_name}'
        return obj


person = Person('John', 'Doe')
print(person.full_name)

print(person.__dict__)
```

```
John Doe
{'first_name': 'John', 'last_name': 'Doe', 'full_name': 'John Doe'}
[Finished in 1.5s]
```