Métodos en Python

Daniela Martínez & José Pinango

29 de marzo de 2022

Tarea 3

(I) Ingresar a la terminal de python (puede ser la terminal que incluye Pycharm).

(II) Escribir

```
>>> aux = [1, 2, 3]
>>> help(type(aux))
```

(III) Explicar para cada uno de los métodos que aparecen qué hacen y dar ejemplos concretos de uso.

Métodos de Listas

1. **___add___(self, value, /)**
   | Return self+value.

   Regresa la concatenación de dos listas.

   Ejemplo:

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

2. **___contains___(self, key, /)**
   Return key in self.

   Indica si un elemento (key) está contenido en una lista.

   Ejemplo:

   ```
    >>> aux
   [1, 3, 3, 7]
   >>> aux.__contains__(8)
   ```

```
False
>>> aux.__contains__(7)
True
>>> 2 in aux
False
```

3. **___delitem___(self, key, /)**
   | Delete self[key].

   Elimina el i-ésimo componente de la lista.

   Ejemplo:

```
>>> l = [1, 2, 3]
>>> l.__delitem__(1)
[1, 3]
>>> del l[1]
[1]
```

4. **___eq___(self, value, /)**
   | Return self==value.

   Indica si una lista (value) es igual a otra.

   Ejemplo:

```
>>> aux
[1, 3, 3, 7]
>>> aux.__eq__([1,2])
False
>>> [1] == aux
False
>>> [1, 3, 3, 7] == aux
True
```

5. **___ge___(self, value, /)**
   | Return self>=value.

   Determina si cada i-ésimo elemento de una lista es mayor o igual que el i-ésimo elemento de otra lista. Específicamente, si el primer elemento de una lista A es igual al primer elemento de otra lista B entonces verifica si el segundo elemento de A es mayor o igual al segundo elemento de B. Si se cumple la igualdad se procede como antes con el siguiente elemento, caso contrario determina True cuando es mayor y False si es menor.

   Ejemplo:

```
>>> l1 = [3, 0, 8, 2]
>>> l2 = [3, 0, 9]
>>> l3 = [4, 0, 9]
>>> l1.__ge__(l2)
False
>>> l2.__ge__(l1)
True
>>> l2.__ge__(l3)
False
>>> l3.__ge__(l2)
True
>>> l1 >= l2
False
>>> l2 >= l1
True
>>> l2 >= l3
False
>>> l3 >= l2
True
```

6. **___getattribute___(self, name, /)**
   | Return getattr(self, name).

   Es un método que determina si la lista tiene el atributo o no. En caso de si tenerlos, retornará la dirección de la memoria donde se encuentra el atributo.

   Ejemplo:

```
>>> aux.__getattribute__('__add__')
<method-wrapper '__add__' of list object at 0x000001C5D5F0AB80>
>>> aux.__getattribute__('sort')
<built-in method sort of list object at 0x000001C5D5F0AB80
>>> aux.__getattribute__('__bool__')
AttributeError: 'list' object has no attribute '__bool__'
```

7. **___getitem___(...)**
   | x.___getitem___(y) <==>x[y]

   Regresa el i-ésimo elemento de la lista.

   Ejemplo:

```
>>> l = [1, 1, 2, 3, 5, 8, 13]
>>> l.__getitem__(4)
5
>>> l[4]
5
```

8. **___gt___(self, value, /)**
   | Return self>value.
   Indica si los elementos de la lista (self) son mayores que los de la lista (value), la comparación comienza a hacerse con el primer elemento de cada lista, en el caso de ser iguales, se compara con el siguiente, y así susecivamente.

   Ejemplo:

```
>>> aux
[1, 3, 3, 7]
>>> aux.__gt__([4,0])
False
>>> aux.__gt__([0,3])
True
>>> aux.__gt__([1,3])
True
>>> aux.__gt__([1,3,3,10])
False
>>> aux>[1,2]
True
```

9. **___iadd___(self, value, /)**
   | Implement self+=value.

   Actualiza la primera lista como la concatenación de ambas listas.

   Ejemplo:

```
>>> l1 = [3, 0, 8, 2]
>>> l2 = [3, 0, 9]
>>> l1.__iadd__(l2)
[3, 0, 8, 2, 3, 0, 9]
>>> l1
[3, 0, 8, 2, 3, 0, 9]
>>> l1.__add__(l2)
[3, 0, 8, 2, 3, 0, 9, 3, 0, 9]
>>> l1
[3, 0, 8, 2, 3, 0, 9]
>>> l1 = [3, 0, 8, 2]
>>> l2 = [3, 0, 9]
>>> l1 += l2
>>> l1
[3, 0, 8, 2, 3, 0, 9]
```

   Observe que el método ___*add*___ regresa la concatenación de ambas listas manteniendo las listas originales sin cambios. En cambio el método

__iadd__ modifica únicamente la primer lista como la concatenación de las dos listas.

10. **__imul__(self, value, /)**
    | Implement self*=value.

    Se crea una nueva lista haciendo una concatenación de la misma lista, el número de veces ingresado (value). Esta nueva lista será guardada en lista invocada.

    Ejemplo:

    ```
    >>> aux
    [1, 3, 3, 7]
    >>> aux.__imul__(2)
    [1, 3, 3, 7, 1, 3, 3, 7]
    >>> aux=[5,6]
    >>> aux*=2
    >>> aux
    [5, 6, 5, 6]
    ```

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

    Sobreescribe a la lista con la nueva lista del argumento.

    Ejemplo:

    ```
    >>> l = [1, 2, 3]
    >>> l.__init__([2, 4, 6, 8, 10])
    >>> l
    [2, 4, 6, 8, 10]
    ```

12. **__iter__(self, /)**
    | Implement iter(self).

    Crea un iterador de listas, que permite imprimir las entradas de la lista en forma iterada.

    Ejemplo:

    ```
    >>> fruits
    ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
    >>> iterator_fruits=iter(fruits)
    >>> print(next(iterator_fruits))
    orange
    >>> print(next(iterator_fruits))
    apple
    ```

```
>>> print(next(iterator_fruits))
pear
>>> print(next(iterator_fruits))
banana
>>> print(next(iterator_fruits))
kiwi
>>> print(next(iterator_fruits))
apple
>>> print(next(iterator_fruits))
banana
```

13. **___le___(self, value, /)**
    | Return self<=value.

    Determina si cada i-ésimo elemento de una lista es menor o igual que el i-ésimo elemento de otra lista. Es similar al método *___ge___* visto anteriormente.

    Ejemplo:

```
>>> l1 = [5, 7, 3]
>>> l2 = [5, 7, 6, 1]
>>> l3 = [4, 7, 6, 1]
>>> l1.__le__(l2)
True
>>> l2.__le__(l1)
False
>>> l2.__le__(l3)
False
>>> l3.__le__(l2)
True
>>> l1 <= l2
True
>>> l2 <= l1
False
>>> l2 <= l3
False
>>> l3 <= l2
True
```

14. **___len___(self, /)**
    | Return len(self).

    Da la longitud de una lista (self), esto es el número de entradas.

    Ejemplo:

```
>>> len([1,2,3])
3
>>> len(['orange','apple','pear'])
3
>>> aux.__len__()
2
```

15. **___lt___(self, value, /)**
    | Return self<value.

    Determina si cada i-ésimo elemento elemento de una lista es menor que el i-ésimo elemento de otra lista.

    Ejemplo:

```
>>> l1 = [3, 1, 7]
>>> l2 = [2, 0, 6, 2]
>>> l3 = [4, 1]
>>> l1.__lt__(l2)
False
>>> l2.__lt__(l1)
True
>>> l2.__lt__(l3)
True
>>> l3.__lt__(l2)
False
>>> l1 < l2
False
>>> l2 < l1
True
>>> l2 < l3
True
>>> l3 < l2
False
```

16. **___mul___(self, value, /)**
    | Return self*value. Se crea una lista haciendo una concatenación de la misma lista, el número de veces ingresado (value), sin embargo está lista no se guarda en la lista invocada.

    Ejemplo:

```
>>> [0,1].__mul__(4)
[0, 1, 0, 1, 0, 1, 0, 1]
>>> [0,1]*(2)
[0, 1, 0, 1]
```

17. **___ne___(self, value, /)**
    | Return self!=value.

    Determina si las listas son distintas.

    Ejemplo:

```
>>> l1 = [2, 4, 5]
>>> l2 = [2, 4, 5]
>>> l3 = [2, 5, 5]
>>> l1.__ne__(l2)
False
>>> l2.__ne__(l3)
True
>>> l1 != l2
False
>>> l2 != l3
True
```

18. **___repr___(self, /)**
    | Return repr(self).

    Cambia el tipo de clase y lo convierte en tipo string.

    Ejemplo:

```
     >>> repr(aux)
'[1, 2, 1, 2]'
     >>> type(repr(aux))
<class 'str'>
     >>> colors=['red','blue']
     >>> colors.__repr__()
"['red', 'blue']"
```

19. **___reversed___(self, /)**
    | Return a reverse iterator over the list.

    Regresa una lista revertida de la lista inicial.

    Ejemplo:

```
>>> l = [1, 1, 2, 3, 5, 8]
>>> r = l.__reversed__()
>>> r
<list_reverseiterator object at 0x7f9fc9ef8e80>
>>> list(r)
[8, 5, 3, 2, 1, 1]
>>> l
```

```
[1, 1, 2, 3, 5, 8]
>>> s = reversed(l)
>>> s
<list_reverseiterator object at 0x7f9fc9f88040>
>>> list(s)
[8, 5, 3, 2, 1, 1]
>>> l
[1, 1, 2, 3, 5, 8]
```

20. **___rmul___(self, value, /)**
    | Return value*self.

    El resultado será una lista concatenada el número de veces (value), pero esta vez la operación comenzará por la derecha.

    Ejemplo:

    ```
    >>> colors
    ['red', 'blue']
    >>> 2*colors
    ['red', 'blue', 'red', 'blue']
    >>> aux
    [1, 2, 3, 4]
    >>> aux.__rmul__(0)
    []
    ```

21. **___setitem___(self, key, value, /)**
    | Set self[key] to value.

    Asigna un objeto en la i-ésima posición de la lista.

    Ejemplo:

    ```
    >>> l = [1, 1, 2, 3, 5, 8]
    >>> l.__setitem__(1, 100)
    >>> l
    [1, 100, 2, 3, 5, 8]
    >>> l.__setitem__(2, [3, 3])
    >>> l
    [1, 100, [3, 3], 3, 5, 8]
    >>> l.__setitem__(3, (9, 6))
    >>> l
    [1, 100, [3, 3], (9, 6), 5, 8]
    >>> l.__setitem__(4, {2, 0})
    >>> l
    [1, 100, [3, 3], (9, 6), {0, 2}, 8]
    >>> l.__setitem__(5, 'Hola Mundo')
    ```

```
>>> l
[1, 100, [3, 3], (9, 6), {0, 2}, 'Hola Mundo']
>>> l[0] = 999
>>> l
[999, 100, [3, 3], (9, 6), {0, 2}, 'Hola Mundo']
```

22. **___sizeof___(self, /)**
    | Return the size of the list in memory, in bytes.

    Retorna el número de bytes usados en la memoria de la maquina para guardar la lista.

    Ejemplo:

```
     >>> lista1
['H', 'o', 'l', 'a', ' ', 'm', 'u', 'n', 'd', 'o']
     >>> lista1.__sizeof__()
120
```

23. **append(self, object, /)**
    | Append object to the end of the list.

    Añade un objeto al final de la lista.

    Ejemplo:

```
>>> l = [1, 2, 4, 5]
>>> l.append(100)
>>> l
[1, 2, 4, 5, 100]
>>> l.append([2, 3])
>>> l
[1, 2, 4, 5, 100, [2, 3]]
>>> l.append('Python')
>>> l
[1, 2, 4, 5, 100, [2, 3], 'Python']
```

24. **clear(self, /)**
    | Remove all items from list.

    Quita todas las entradas de la lista, y la lista quedara guardada como vacía.

    Ejemplo:

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

```
>>> aux
[]
```

25. **copy(self, /)**
    | Return a shallow copy of the list.

    Regresa una copia superficial de la lista.

    Ejemplo:

```
>>> l = [1, 2, 3]
>>> c = l.copy()
>>> c
[1, 2, 3]
>>> c.__setitem__(1, 100)
>>> c
[1, 100, 3]
>>> l
[1, 2, 3]
```

26. **count(self, value, /)**
    | Return number of occurrences of value.

    Retorna el número de apariciones de una entrada en una lista.

    Ejemplo:

```
>>> aux=[0,1,0,0,0,1]
>>> aux.count(0)
4
>>> aux.count(1)
2
>>> colors
['red', 'blue']
>>> colors.count('blue')
1
```

27. **extend(self, iterable, /)**
    | Extend list by appending elements from the iterable.

    Une una lista a otra.

    Ejemplo:

```
>>> l1 = [2, 4, 6]
>>> l2 = [1, 3, 5]
>>> l1.extend(l2)
```

```
>>> l1
[2, 4, 6, 1, 3, 5]
```

Observe que cumple la misma función que ___*iadd*___.

28. **index(self, value, start=0, stop=9223372036854775807, /)**
    | Return first index of value.
    |
    | Raises ValueError if the value is not present.

    Encuentra el valor de la entrada donde aparece por primera vez el valor (value) de interes. En casi de no aparecer, lo notificará.

    Ejemplo:

    ```
    >>> fruits
    ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
    >>> fruits.index('kiwi')
    4
    >>> fruits.index('blueberry')
    ValueError: 'blueberry' is not in list
    ```

29. **insert(self, index, object, /)**
    | Insert object before index.

    Añade un objeto en un índice específico.

    Ejemplo:

    ```
    >>> l.insert(4, 100)
    >>> l
    [1, 1, 2, 3, 100, 5]
    # Añade un objeto en la penúltima posición
    >>> l.insert(-1, 10)
    >>> l
    [1, 1, 2, 3, 100, 10, 5]
    # Añade un objeto en la última posición
    >>> l.insert(len(l), 999)
    >>> l
    [1, 1, 2, 3, 100, 100, 5, 999]
    # Si el índice sobrepasa la longitud de la lista,
    # el objeto se añade en la última posición
    >>> l.insert(1000, 99)
    >>> l
    [1, 1, 2, 3, 100, 100, 5, 999, 99]
    >>> l.insert(1000, [1, 2])
    >>> l
    [1, 1, 2, 3, 100, 100, 5, 999, 999, [1, 2]]
    ```

```
>>> l.insert(1000, 'Hello')
>>> l
[1, 1, 2, 3, 100, 100, 5, 999, 999, [1, 2], 'Hello']
```

30. **pop(self, index=-1, /)**
    | Remove and return item at index (default last).
    |
    | Raises IndexError if list is empty or index is out of range.

Remueve la entrada de una lista eligiendo el indice de la que se desea eliminar, además retorna la entrada eliminada.

Ejemplo:

```
>>> aux=[1,2,3,4]
>>> aux.pop(0)
1
>>> aux
[2, 3, 4]
>>> list_empty=[]
>>> list_empty.pop(0)
IndexError: pop from empty list
```

31. **remove(self, value, /)**
    | Remove first occurrence of value.
    | Raises ValueError if the value is not present.

Elimina el primer elemento de la lista que coincida con el argumento.

Ejemplo:

```
>>> l = [5, 4, 2, 2, 2, 6, 6, 7, 9]
>>> l.remove(6)
>>> l
[5, 4, 2, 2, 2, 6, 7, 9]
>>> l.remove(7)
>>> l
[5, 4, 2, 2, 2, 6, 9]
>>> l = [1, 2, 'Python']
>>> l.remove('Python')
>>> l
[1, 2]
```

32. **reverse(self, /)**
    | Reverse *IN PLACE*.

Reversa los elementos de la lista basandose en el indice, el último comenzará en el primer lugar e irá descendiendo. **Ejemplo:**

```
>>> aux=[1,2,3,4]
>>> aux.reverse()
>>> aux
[4, 3, 2, 1]
>>> colors=['red','blue','white','black']
>>> colors.reverse()
>>> colors
['black', 'white', 'blue', 'red']
```

33. **sort(self, /, *, key=None, reverse=False)**
    | Sort the list in ascending order and return None.
    | The sort is in-place (i.e. the list itself is modified) and stable
    | (i.e. the order of two equal elements is maintained).
    | If a key function is given, apply it once to each list item and
    | sort them, ascending or descending, according to their
    | function values.
    | The reverse flag can be set to sort in descending order.

    Ordena los elementos de la lista de menor a mayor. Usando el argumento $reverse = True$ se obtiene el ordenamiento al revés.

    Ejemplo:

    ```
    >>> l = [3, -8, 1, 3, 2, -1, 10]
    >>> l.sort()
    >>> l
    [-8, -1, 1, 2, 3, 3, 10]
    >>> r = [-4, 2, 4, -4, 3, 9]
    >>> r.sort(reverse = True)
    >>> r
    [9, 4, 3, 2, -4, -4]
    ```

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

    Crea una lista nueva, indicado el tipo de formato que se desee.

    Ejemplo:

    ```
    >>> list_example=list.__new__(list)
    >>> list_example
    []
    ```

35. **Data and other attributes defined here:**
    ___hash___ = None.

    No realiza ningún cambio a la lista.

Ejemplo:

```
>>> l.__hash__
>>> l
[-8, -1, 1, 2, 3, 3, 10]
```