IWI 131- PROGRAMACIÓN

Ayudantía 9

Ayudante: Anastasiia Fedorova

Paralelo: 212

Fecha: 10.06.2020

RECORDANDO LO APRENDIDO

Ya hemos visto los strings – unas cadenas de caracteres, con acceso por índice (positivo o negativo) y operaciones relacionadas (por ejemplo, concatenar).

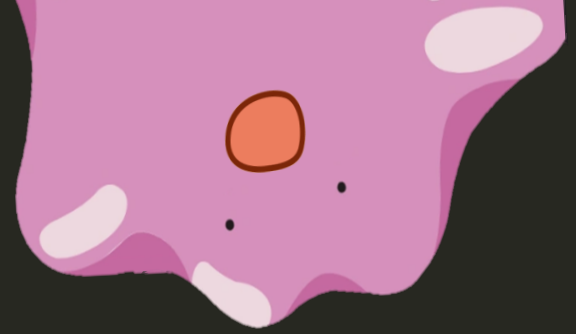
Notamos podemos guardar muchas cosas en un string - por ejemplo, strings CSV (comma separated values) pueden contener muchos valores, separados por el delimitador coma.

Pero no es lo más óptimo, pués de algún modo tenemos que sacar los valores de string, y no tenemos cómo almacenarlos para usarlos después – o no sabíamos, hasta ahora. Aquí es cuando aparecen las listas, herramienta muy poderosa de cual dispone Python.

```
date,location,cases
2013-11-05,United States,4
2013-11-05,Germany,8
2013-11-11,South Africa,9
2013-11-12,Japan,6
```

Ejemplo de string de tipo CSV
Cada string contiene fecha, lugar y cantidad de casos de brotse de cierta enfermedad.

¿STRINGS != LISTAS?



Aunque comparten mucho, los strings no son listas según la definición. Sin embargo, conocer cómo se usan los strings nos va a ayudar mucho para comprender las listas.

Similitudes:

Usan índices

Usan slices

Son colecciones

Diferencias:

Strings son colecciones de SOLO caracteres

Listas pueden contener otros elementos (incluso otras listas)

Listas son mutables (se puede cambiar sus elementos)

CREACIÓN DE LISTA

Listas se anotan con corchetes – []. Se puede crear una lista vacía en Python de dos maneras:

```
lista = []
```

```
lista = list()
```

Además, podemos crear una lista con elementos ya definidos, simplemente encerrándolos en corchetes. Los elementos no necesariamente tienen que ser de mismo tipo – en Python, las listas son heterogéneas.

```
z = [3, 7, 4, 2]
```

z =	[3,	7,	4,	2]
index	0	1	2	3

ACCESO CON ÍNDICES

Para acceder a algún elemento de nuestra lista, utilizaremos los índices, al igual que con los strings – `lista[indice]`.

```
z = [3, 7, 4, 2]
```

z =	[3,	7,	4,	**2**]
index	0	1	2	3
negative index	-4	-3	-2	-1

Ilustración de los índices de la lista z.

SLICES

Los slices en las listas sirven para obtener unos subconjuntos de la lista. Funcionan de la misma manera que en strings – `lista[inicio:fin:paso(opcional)]`

z =	[3,	7,	4,	2]
index	0	1	2	3

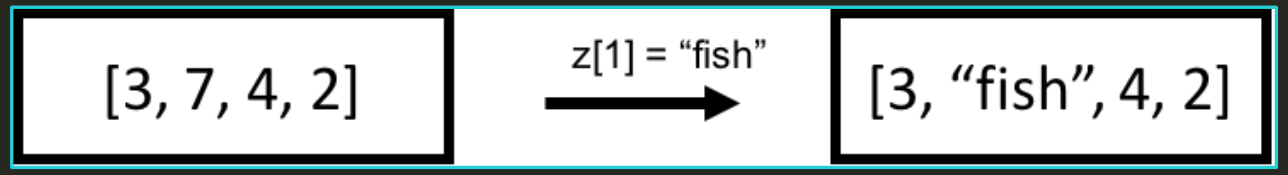
Ilustración de slice de 0 a 2 – en azul se anotan los elementos de la sublista (slice).

```
z = [3, 7, 4, 2]
print(z[0:2])
```

CAMBIAR UN VALOR EN LISTA

Dijimos antes que las listas son **mutables** – pues, se puede cambiar cualquier elemento de la lista, lo que no es posible para los strings (no podemos cambiar las letras de una palabra).

En listas basta nombrar el índice en el cual queremos cambiar un elemento y dar el nuevo valor en la asignación:



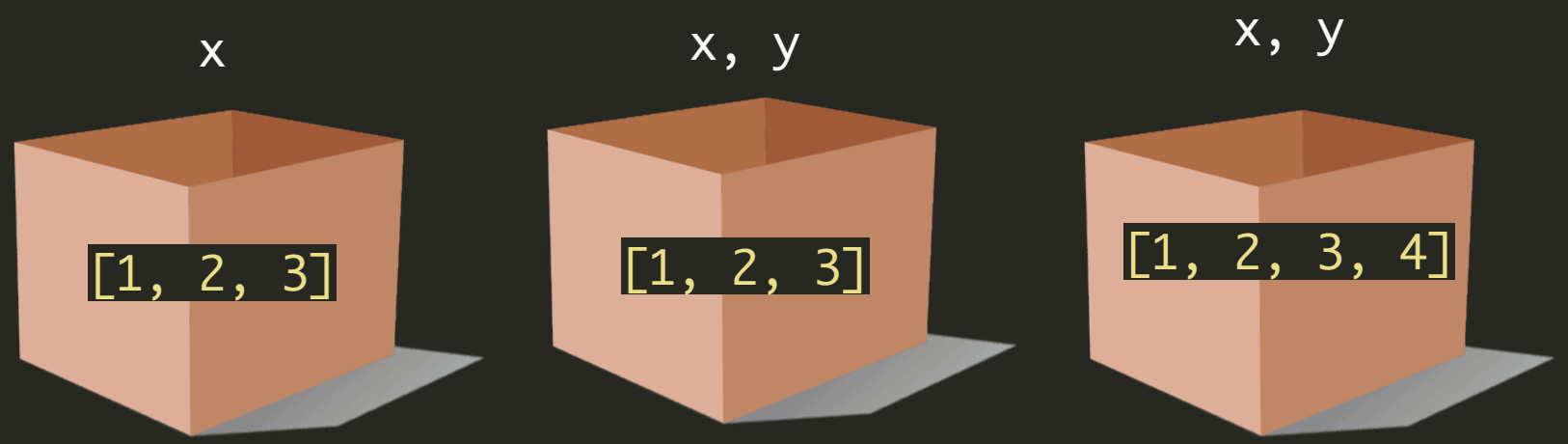
Cambio el elemento de indice 1 (el valor 7) por el string “fish”.

```
z = [3, 7, 4, 2]
z[1] = "fish"
```

REASIGNACIÓN DE LISTAS

Al igualar una variable a otra, donde la segunda contiene una lista, al modificar una de las variables (cualquiera) **se modifica la lista original**.

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



Cambio de ¨etiqueta¨ de la caja – es equivalente llamarla tanto y como x.

En este sentido, podemos decir que ahora la caja de x tambien se puede referirse con el nombre y. Es decir, x e y son variables equivalentes.

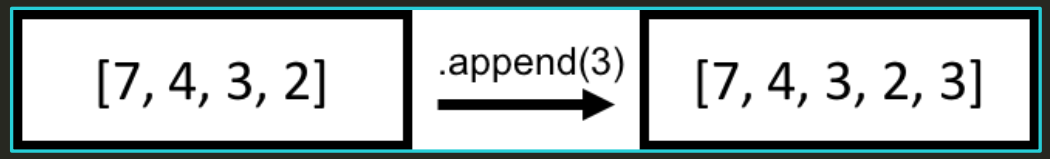
MÉTODOS EN LISTAS

Los métodos son unas funciones, las cuales nos permites trabajar con la estructura. En caso de la lista, los métodos más importantes son:

- `.append(valor)` – agrega un elemento al final de la lista
- `.sort()` – ordena la lista de **menor a mayor**
- `.sort(reverse=True)` – ordena de **mayor a menor**
- `.count(valor)` – cuenta la cantidad de repeticiones de un valor en la lista
- `.index(valor)` – retorna el índice de la primera ocurrencia del valor
- `.insert(pos, valor)` – inserta el valor en la posición dada de la lista

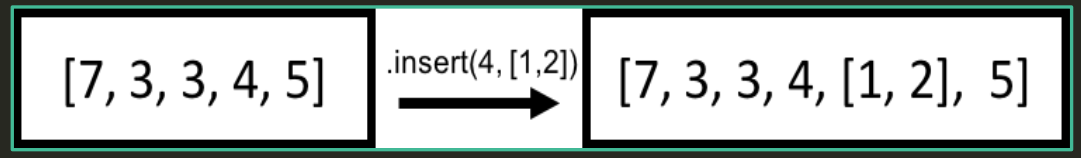
INSERCIÓN

`.append(valor)` –
agrega elemento al final de la lista




```
z = [7, 4, 3, 2]
z.append(3)
```

`.insert(pos, valor)`
inserta elemento en posición pos




```
z = [7, 3, 3, 4, 5]
z.insert(4, [1, 2])
```

EJERCICIO 1

Un alumno de la USM, al dar prueba online se encontró con un problema – por algúna razón, sus respuestas a las preguntas de certamen no se marcaron como enviadas. El alumno, superado por la frustración, procedió a clickear muchas veces sobre el botón enviar, por lo que algunas de sus respuestas se registraron más de una vez.

Para ayudar a los profesores en la revisión, se les pide escribir una función `borrar_repeticiones(l)`, la cual recibe una lista con posibles valores repetidos y debe retornar una nueva lista sin repeticiones.

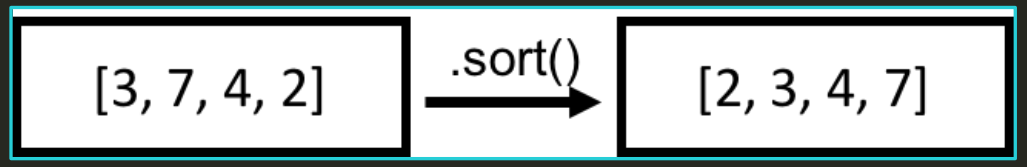

```
l = [1, 1, 1, 2]
print(borrar_repeticiones(l))
>>> [1, 2]
```

SOLUCIÓN

```
def borrar_repeticiones(l):
    sin_rep = []
    for i in l:
        if i not in sin_rep:
            sin_rep.append(i)
    return sin_rep
```

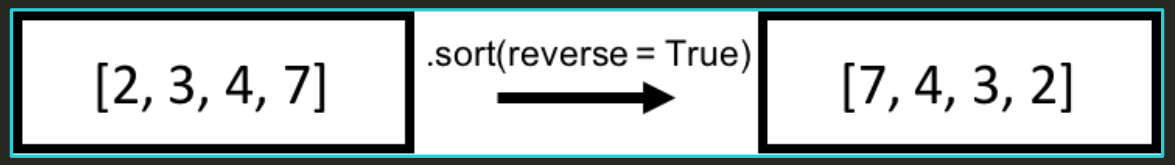
ORDENAMIENTO

`.sort()` - ordena la lista de **menor** a **mayor**




```
z = [7, 4, 3, 2]
z.sort()
```

`.sort(reverse=True)` ordena de la lista **mayor** a **menor**




```
z = [2, 3, 4, 7]
z.sort(reverse=True)
```

EJERCICIO 2.1

Una de las alumnas de la USM tiene una extensa colección de libros, y estos los marca con un cierto número entero. De este modo, ella puede saber cuantos libros tiene en su posesión en total. Se le ocurrió ordenar sus estanterías por el número, que asignó a cada libro – pero cuando vió sus libros, se dio cuenta que algunos de ellos ya están ordenados.

Crear una función que retorna True si la lista original de enteros está ordenada. Avisar al usuario por consola que el orden de la lista – de menor a mayor o de mayor a menor. Si la lista no está ordenada, retornar False. Asuma que la lista vacía no tiene orden.

```
print(ordenado([1, 2, 3, 4, 5]))
print(ordenado([]))
print(ordenado([1, 0, 9, 5]))
```

```
Lista esta ordenada de menor a mayor
True
False
False
```

SOLUCIÓN

```
def ordenado(l):
    if l == []:
        return False
    # no podemos hacer copylist = l, porque al hacer sort, se cambiaran ambas listas
    copylist = list(l)
    # no podemos hacer copylist = copylist.sort()
    # pues sort no tiene retorno - solo cambia la copylist
    copylist.sort()
    if l == copylist:
        print("Lista esta ordenada de menor a mayor")
        return True
    else:
        copylist.sort(reverse=True)
        if l == copylist:
            print("Lista esta ordenada de mayor a menor")
            return True
        else:
            return False
```

ORDENAMIENTO

A veces necesitamos ordenar los elementos de acuerdo a otro criterio. Para esto, el método `sort` acepta un parámetro con nombre llamado `key`, que debe ser una función que asocia a cada elemento el valor que será usado para ordenar.

```
def negativo(x):
    return -x
a = [6, 1, 4, 0, 9]
a.sort(key=negativo)

>>> [9, 6, 4, 1, 0]
```

La lista es ordenada comparando los negativos de sus elementos, aunque son los elementos originales los que aparecen en el resultado

EJERCICIO 2.2

Crear las funciones, que permiten:

A) Ordenar una lista de strings de la más corta a la más larga

```
animales = ['conejo', 'ornitorrinco', 'pez', 'hipopotamo', 'tigre']
>>> ['pez', 'tigre', 'conejo', 'hipopotamo', 'ornitorrinco']
```

SOLUCIÓN

```
def largo(palabra):
    return len(palabra)


animales = ['conejo', 'ornitorrinco', 'pez', 'hipopotamo', 'tigre']
animales.sort(key=largo)
print(animales)
```

EJERCICIO 2

Crear las funciones, que permiten:

B) Ordenar una lista de listas según la suma de sus elementos, de menor a mayor

```
a = [[6, 1, 5, 9], [0, 0, 4, 0, 1], [3, 2, 12, 1], [1000],[7, 6, 1, 0]]
>>> [[0, 0, 4, 0, 1], [7, 6, 1, 0], [3, 2, 12, 1], [6, 1, 5, 9], [1000]]
```

SOLUCIÓN

```
def suma(lista):
    suma = 0
    for i in lista:
        suma += i
    return suma

a = [[6, 1, 5, 9], [0, 0, 4, 0, 1], [3, 2, 12, 1], [1000],[7, 6, 1, 0],]
a.sort(key=suma)
print(a)
```

EJERCICIO 2

Crear las funciones, que permiten:

C) Hacer que queden los números impares a la izquierda y los pares a la derecha

```
l = [55, 222, 47, 81, 82, 44, 218, 82, 20, 96, 82, 251]
>>> [55, 47, 81, 251, 222, 82, 44, 218, 82, 20, 96, 82]
```

SOLUCIÓN

```
def paridad(elem):
    return elem % 2

l = [55, 222, 47, 81, 82, 44, 218, 82, 20, 96, 82, 251]
l.sort(key=paridad, reverse=True)
print(l)
```

.COUNT(VALOR)

El método `count` permite obtener la cantidad de repeticiones de cierto elemento en la lista:

```
random_list = [4, 1, 5, 4, 10, 4]
random_list.count(5)
```

```
print(random_list.count(5))
```

```
1
```

Al imprimir el retorno del método count, obtenemos el valor entero. En este caso, 5 en la lista aparece solo 1 vez.

.INDEX(VALOR)

Retorna el índice de la primera apariencia de cierto valor en la lista. Además, se puede especificar desde qué posición se inicia la busqueda:

z =	[4,	1,	5,	4,	10,	4]
index	0	1	2	3	4	5

```
print(z.index(4))
```

```
print(z.index(4))
0
```

z =	[4,	1,	5,	4,	10,	4]
index	0	1	2	3	4	5

```
print(z.index(4, 3))
```

```
print(z.index(4, 3))
3
```

EJERCICIO

Dados los nombres de N alumnos de IWI131 y sus puntajes en Certamen 1, se les pide guardarlos en una lista de listas de la forma `[[nombre1,puntaje1], [nombre2, puntaje2]...]`.

Imprima el/los nombres de alumno/a(s) con la *segunda peor nota*.

La primera linea de input contiene el numero N de alumno/as, donde las 2N siguientes lineas contienen el nombre y el puntaje de cada uno de esto/as.

Input:
5
Javier
67.21
Luna
67.21
Victor
67.2
Maria
71
Gonzalo
69

Output:
Javier
Luna

SOLUCIÓN

```
def get_score(l):
    return l[1]

def get_name(l):
    return l[0]

def borrar_repeticiones(l):
    list_set = []
    for i in l:
        if i[1] not in list_set:
            list_set.append(i[1])
    return list_set

def print_list(l):
    for i in l:
        print(i)
```

SOLUCIÓN

```
# creacion de lista de listas, segun el
enunciado
ll = []
n = int(input())
for x in range(n):
    name = input()
    score = float(input())
    ll.append([name, score])
# ordeno por el puntaje, de menor a mayor
ll.sort(key=get_score)

answer = []
# obtengo lista con puntajes unicos, sin
repeticiones
s = set_simulator(ll)
```

```
# s[1] es el segundo minimo
# si len(s) = 1, todos
# obtuvieron el mismo puntaje y
# por ello, no hay segundo minimo.
if len(s) > 1:
    for li in ll:
        if get_score(li) == s[1]:
            answer.append(li[0])
answer.sort()
print_list(answer)
```