Python Lists & Tuples

Cheat Sheet

Listas

Una **lista** es una estructura de datos en Python que representa una secuencia **ordenada** de elementos que es **mutable** o **cambiable**. Cada elemento o valor dentro de una lista se llama **ítem**. Así como las cadenas (strings) se definen como caracteres entre comillas, las listas se definen por tener valores entre **corchetes [ ].**

```
> [10,True,"Hola",9.4,[1,2],[],41]
```

Tupla

Una **tupla** en Python es una estructura de datos que representa una **secuencia ordenada** de elementos, pero a diferencia de las listas, **es inmutable**, lo que significa que **no se puede modificar** después de haber sido creada (no se pueden agregar, eliminar ni cambiar sus elementos).

Las tuplas se definen **entre paréntesis** () y los elementos están separados por comas.

```
> (10,True,"Hola",9.4,[1,2],[],41)
```

Sintaxis sobre listas y tuplas

L = [1, 2, 3, 4, 5]	Listas se crean con[ ]
T = (10, 20, 30, 40, 50)	Tuplas se crean con ( )
L[0]	Retorna 1er item de *L* (1)
T[0]	Retorna 1er item de *T* (10)
L[1:4]	**Slicing**: Retorna 2do al 4to item de *L* ([2, 3, 4]). Los slice son listas. El rango del Slice es abierto a la derecha, no incluye ese elemento
T[1:4]	**Slicing**: Retorna 2do al 4to item de *T* ((20, 30, 40))
L[-1]	Retorna el último elemento de L (5). Los índices negativos, iniciando en -1, recorren la lista de derecha a izquierda.
T[-1]	Retorna el último elemento de T (50).
L[-3:-1]	Retorna [3,4]. Slicing con negativos funciona de igual manera, abierto a la derecha.
T[-3:-1]	Retorna [30,40]
L[1] = 22	Asigna *22* al 2do elemento de *L* (L == [1, 22, 3, 4, 5])
T[1] = 22	ERROR: no puede modificar las tuplas
L[0:2] = [11, 22]	Asigna *11* y *22* al 1ro y 2do elementos de *L* (L == [11, 22, 3, 4, 5])

Métodos sobre listas

a = ['a', 'b', 'c']

b = [1, 3, 2]

a + b	Retorna *a* concatenado con *b* (['a', 'b', 'c', 1, 3, 2])
'c' in a	Devuelve **True** si 'c' está en la lista a, y **False** en caso contrario (**True**)
len(a)	Devuelve la cantidad de elementos en a (**3**)
a.append('d')	Agrega 'd' al final de la lista a (**a == ['a', 'b', 'c', 'd']**)
a.extend(['d', 'e', 'f'])	Agrega cada elemento del iterable al final de a (**a == ['a', 'b', 'c', 'd', 'e', 'f']**)
a.insert(1, 'd')	Inserta 'd' en la posición 1 de a (**a == ['a', 'd', 'b', 'c']**)
a.pop()	Devuelve el último elemento de la lista y lo elimina (**'c'**)
a.pop(1)	Devuelve el segundo elemento de a y lo elimina (**'b'**)
a.remove('b')	Elimina la primera ocurrencia de 'b' en a (**a == ['a', 'c']**)
a.clear()	Limpia completamente la lista (**a == []**)
a.index('b')	Devuelve el índice de la primera ocurrencia de 'b' (**1**)
a.count('b')	Devuelve el número de veces que 'b' aparece en a (**1**)
b.sort()	Devuelve una versión ordenada de b (**[1, 2, 3]**)
a.reverse()	Invierte el orden de la lista a (**['c', 'b', 'a']**)
a.copy()	Devuelve una copia de a

Métodos sobre tuplas

t1 = ('a', 'b', 'c')

t2 = (1, 2, 3)

t1 + t2	Devuelve una versión concatenada de t1 y t2
2 in t2	Devuelve **True** si 2 está en t2, y **False** en caso contrario (**True**)
len(t1)	Devuelve la cantidad de elementos en t1 (**3**)
t2.count(2)	Devuelve el número de veces que 2 aparece en t2 (**1**)
t2.index(1)	Devuelve el índice de la primera ocurrencia de 1 (**0**)
t1 + t2	Devuelve una versión concatenada de t1 y t2

Python Lists & Tuples

Cheat Sheet

Slicing para Loops

El slicing retorna una porción de la lista. Se utiliza mediante rangos de índices de la lista con el formato [inicio : final], siempre abierto en el final, es decir, no incluye el elemento en índice final.

```
lista = [inicio : final]
```

Cuando uno de los extremos del slice se deja en blanco, incluye desde el inicio o hasta el final.

```
l1 = [ i : ]    # incluye de i hasta el
                  final inclusive
l1 = [ : n ]   # incluye desde inicio
                  hasta n abierto (sin n)
```

l1 = [10, 20, 30, 40, 50]

l1 [0 :]	Retorna toda la lista, desde 0 hasta el final
l1 [: 5]	Retorna toda la lista, desde el inicio hasta el índice 4, que para l1 es el final (50)
l1 [: len(l1)]	Retorna toda la lista, desde el inicio hasta len(l1) == 5 abierto
l1 [1 :]	Retorna la lista sin el primer elemento. **Quita el primer elemento de la lista**.
l1 [: -1]	Retorna la lista sin el último elemento. **Quita el último elemento**
l1 [: len(l1)-1]	Retorna la lista sin el último elemento. **Quita el último elemento**

Loop 1: For each

El for es un bucle de Python para el desarrollo iterativo. El for está dirigido a recorrer estructuras de datos iterables: *secuencias como listas.*

```
for elemento in lista:
      # cada iteración elemento vale cada
        item de la lista, iniciando en el
        primer elemento y termina SOLO
        hasta llegar al último item
```

- ✓ El for anterior se conoce como **For Each** (para cada) porque recorre cada elemento de la lista.
- ✓ La variable `elemento` puede tener cualquier nombre, se recomienda que sea acorde a los valores de la lista, si está recorriendo una lista de estudiantes el nombre recomendado sería `estudiante`.
- ✓ Los for terminan **solos** al llegar al final de la lista, es decir, cuando ha recorrido todos los elementos de la lista dada.

Loop 2: For i

El for puede hacerse sobre rangos de números, utilizando la función **range**(n).
La función range crea una estructura iterable (no es lista) y es un rango abierto a la derecha, es decir, no incluye el valor final del rango. Ejemplo:

```
range ( 5 )    es 0, 1, 2, 3, 4
range (2, 5)   es 2, 3, 4
```

Es común y muchas veces útil recorrer una lista por sus índices. Por ejemplo, si se tiene la lista:

l1 = [10, 20, 30, 40, 50]

Se puede recorrer cada uno de los items **l1[i]**, para lo cual se requiere que i tenga los valores de los índices desde 0 hasta 4, los índices de los elementos de la lista l1. Estos índices se pueden obtener con la función **range** (**len**(l1)).

```
for i in range(len(lista)):
      elemento = lista [i]
      # cada iteración i vale cada valor
        del rango, iniciando en 0 y
        terminado en el largo-1 porque es
        abierto, no incluye el índice
        largo. Por tanto, lista[i] será
        cada elemento.
```

- ✓ El for anterior se conoce como **For i** (i es index) porque recorre la lista utilizando índices.
- ✓ La variable `i` es una convención para índices. En caso de tener for anidados con índices, la variable del primer for será `i`, el del segundo `j`, el tercero `k`.
- ✓ Los for con índices terminan **solos** al llegar al final del recorrido del rango definido.

Por Diego Mora