Introducción a las ciencias de la computación
y programación en Python

Tuplas y listas: mutabilidad y clonado

Rodrigo Chang

Banco de Guatemala

Rodrigo Chang <rrcp@banguat.gob.gt>

Abstract

"La simplicidad es un prerrequisito para la confiabilidad"
— Edsger W. Dijkstra

Veremos dos tipos nuevos compuestos en Python. Hablaremos de los conceptos de *alias*, mutabilidad y clonado.

Tuplas

- Secuencia ordenada de elementos, que pueden ser de diferentes tipos.
- No es posible alterar sus elementos, son **inmutables**.
- Se representan con paréntesis

```
# Tupla vacia
te = ()
# Tupla de ejemplo
t = (2, "pes", 3.2)

t[0]       # devuelve 2
t + (5,6) # devuelve (2, "pes" 3.2, 5, 6)
t[1:2]     # slice, devuelve ("pes", )
t[1:3]     # slice, devuelve ("pes", 3.2)
len(t)     # devuelve 3

t[1] = 4 # error, inmutable
```

Tuplas

- conveniently used to **swap** variable values

```
x = y
y = x
```

```
temp = x
x = y
y = temp
```

```
(x, y) = (y, x)
```

- used to **return more than one value** from a function

```
def quotient_and_remainder(x, y):
    q = x // y
    r = x % y
    return (q, r)
(quot, rem) = quotient_and_remainder(4,5)
```

integer division

Iteración sobre tuplas

- Es posible iterar sobre los elementos de una tupla:

```
t = (2, "pes", 3.2)

for elem in t:
    print(elem, "es de tipo", type(elem))
```

Tuplas de tuplas

```
def get_data(aTuple):
    nums= ()
    words = ()
    for t in aTuple:
        nums= nums+ (t[0],)
        if t[1] not in words:
            words = words + (t[1],)

    min_n = min(nums)
    max_n = max(nums)
    unique_words = len(words)
    return (min_n, max_n, unique_words)

aTuple = ((1, 'a'), (2, 'b'), (3, 'c'))
print(get_data(aTuple))
```

Listas

- **Secuencia ordenada** de elementos, accesibles a través de un índice.
- Se denotan utilizando corchetes, `[ ]`
- Los elementos:
  - Son usualmente homogéneos
  - Pero pueden ser de diferentes tipos (práctica poco común, pero válida).
- Los elementos pueden ser **alterados**, por lo que la lista es **mutable**.

Lista e índices

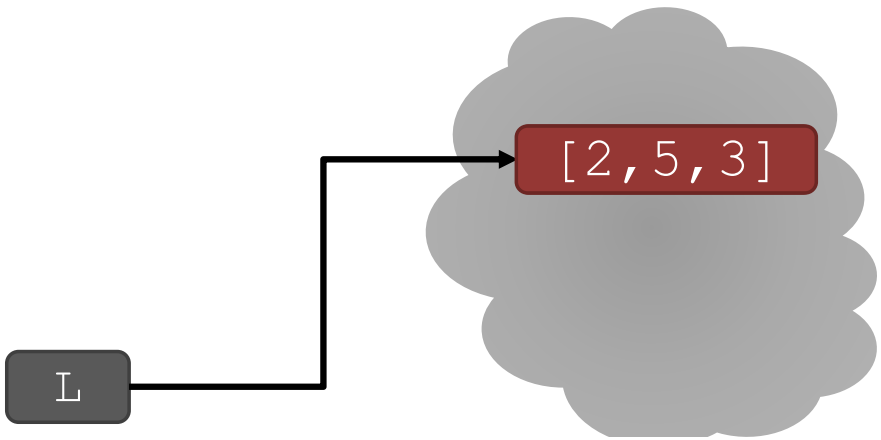
```
L = []
L = [2, 'a', 4, [1,2]]

len(L) # evaluates to 4
L[0]   # evaluates to 2
L[2]+1 # evaluates to 5
L[3]   # evaluates to [1,2], another list!
L[4]   # gives an error

i = 2
L[i-1] # evaluates to 'a' since L[1]='a' above
```

Cambiando elementos

- lists are **mutable**!
- assigning to an element at an index changes the value

```
L = [2, 1, 3]
L[1] = 5
```

- `L` is now `[2, 5, 3]`, note this is the **same object** `L`



Iterando sobre una lista

- compute the **sum of elements** of a list
- common pattern, iterate over list elements

```
total = 0
  for i in range(len(L)):
      total += L[i]
  print total
```

```
total = 0
  for i in L:
      total += i
  print total
```

like strings, can iterate over list elements directly

- notice
  - list elements are indexed `0` to `len(L)-1`
  - `range(n)` goes from `0` to `n-1`

Operaciones sobre listas

- **add** elements to end of list with `L.append(element)`
- **mutates** the list!

```
L = [2,1,3]
L.append(5)
```

→ L is now `[2,1,3,5]`

- what is the dot?
  - lists are Python objects, everything in Python is an object
  - objects have data
  - objects have methods and functions
  - access this information by `object_name.do_something()`
  - will learn more about these later

Operaciones sobre listas

- to combine lists together use **concatenation**, + operator, to give you a new list
- **mutate** list with `L.extend(some_list)`

```
L1 = [2,1,3]
L2 = [4,5,6]
L3 = L1 + L2
```

→ `L3` is `[2,1,3,4,5,6]`
`L1, L2` unchanged

```
L1.extend([0,6])
```

→ mutated `L1` to `[2,1,3,0,6]`

Operaciones sobre listas

- delete element at a **specific index** with `del(L[index])`
- remove element at **end of list** with `L.pop()`, returns the removed element
- remove a **specific element** with `L.remove(element)`
  - looks for the element and removes it
  - if element occurs multiple times, removes first occurrence
  - if element not in list, gives an error

all these operations mutate the list

```
L = [2,1,3,6,3,7,0] # do below in order
L.remove(2) → mutates L = [1,3,6,3,7,0]
L.remove(3) → mutates L = [1,6,3,7,0]
del(L[1])   → mutates L = [1,3,7,0]
L.pop()     → returns 0 and mutates L = [1,3,7]
```

Convertir listas a cadenas de texto

- convert **string to list** with `list(s)`, returns a list with every character from `s` an element in `L`
- can use `s.split()`, to **split a string on a character** parameter, splits on spaces if called without a parameter
- use `''.join(L)` to turn a **list of characters into a string**, can give a character in quotes to add char between every element

```
s = "I<3 cs"          → s is a string
list(s)               → returns ['I','<','3',' ','c','s']
s.split('<')          → returns ['I', '3 cs']
L = ['a','b','c']     → L is a list
''.join(L)            → returns "abc"
'_'.join(L)           → returns "a_b_c"
```

Otras operaciones sobre listas

- Funciones para ordenar: `sort()` y `sorted()`
- Operación de reversa: `reverse()`
- Otras operaciones más: `https://docs.python.org/3/tutorial/datastructures.html`

¡Ojo! algunas son funciones y otras son métodos.

```
L = [9,6,0,3]
sorted(L)   # devuelve lista ordenada sin cambiar L
L.sort()    # cambia L = [0, 3, 6, 9]
L.reverse() # cambia L = [9, 6, 3, 0]
```

Listas en memoria

- Las listas son objetos mutables.
- Se comportan de forma diferente a los tipos inmutables.
- Representan objetos en memoria.
- Distintos nombres de variables pueden apuntar al mismo objeto en memoria.
- Cualquier variable que **apunte** a un objeto es afectada.

Cuando trabajemos con lista debemos cuidarnos de los efectos secundarios.

Alias para listas

- `hot` es un alias para `warm`: al cambiar uno, cambia el otro.
- La función `append()` tiene un efecto secundario.

```
a = 1
b = a
print(a)
print(b)

warm = ['red', 'yellow', 'orange']
hot = warm
hot.append('pink')
print(hot)
print(warm)
```

- Ver en Python Tutor

Clonando una lista

- Para crear una nueva lista podemos **copiar cada elemento** utilizando *slicing*.
- Ver en Python Tutor

```
cool = ['blue', 'green', 'grey']
chill = cool[:]
chill.append('black')
print(chill)
print(cool)
```

Ordenando listas

- Utilizar `sort()` altera la lista, devuelve `None`.
- Utilizar `sorted()` no altera la lista, devuelve la lista ordenada.

```
warm = ['red', 'yellow', 'orange']
sortedwarm = warm.sort()
print(warm)
print(sortedwarm)

cool = ['grey', 'green', 'blue']
sortedcool = sorted(cool)
print(cool)
print(sortedcool)
```

- Ver en Python Tutor

Listas de listas de listas de ...

- Es posible **agrupar** una lista dentro de otra.
- Cuidado con los **efectos secundarios**.

```
warm = ['yellow', 'orange']
hot = ['red']
brightcolors = [warm]
brightcolors.append(hot)
print(brightcolors)

hot.append('pink')
print(hot)
print(brightcolors)
```

- Ver en Python Tutor

- Como **buena práctica**, es mejor evitar mutar una lista al iterar sobre ella. Veamos un ejemplo:
- Ver en Python Tutor
- ¿Por qué `L1 = [2,3,4]` y no `[3,4]`? ¿Cómo se puede corregir este código?

```
def remove_dups(L1, L2):
    for e in L1:
        if e in L2:
            L1.remove(e)

L1 = [1, 2, 3, 4]
L2 = [1, 2, 5, 6]
remove_dups(L1, L2)
print(L1, L2)
```

- Solución.

```
def remove_dups_new(L1, L2):
    L1_copy = L1[:]
    for e in L1_copy:
        if e in L2:
            L1.remove(e)

L1 = [1, 2, 3, 4]
L2 = [1, 2, 5, 6]
remove_dups_new(L1, L2)
print(L1, L2)
```

Ejercicios