Programación II

Tuplas

Lic. Federico Severino Guimpel
LCC - FCEIA - UNR

Tópicos

- Tuplas
  - Definición
  - Operador [ ]
- Iteración sobre tuplas
- Tuplas vs Listas
- Tuplas y Listas

Tuplas

Una tupla es una colección ordenada de elementos. Esta definición es similar a la que vimos para Listas. Sin embargo, notaremos que hay diferencias entre ellas.

Para definir una tupla se usan los paréntesis y, separados por comas, los valores que queremos incluir en la misma:

```
>>> t = ('pan', 'huevos', 100, 1234)
>>> t
('pan', 'huevos', 100, 1234)
```

Tuplas

Formalmente el constructor de la tupla es la coma, no el paréntesis, pero el intérprete muestra los paréntesis, y nosotros deberíamos utilizarlos, por claridad.

```
>>> t = 'pan', 'huevos', 100, 1234
>>> t
('pan', 'huevos', 100, 1234)
```

Tuplas

Supongamos que queremos definir una tupla con un único elemento:

```
>>> t = (1)
>>> t
1
```

Como podemos ver, por más que usemos los paréntesis, no estamos definiendo una tupla. La forma correcta de hacerlo sería:

```
>>> t = (1,)
>>> t
(1,)
```

Tuplas

Para referirnos a elementos de una tupla, como en una lista, se usa el operador [ ] con la misma características que las vistas en listas:

```
>>> t = ('pan', 'huevos', 100, 1234)
>>> t[-1]
1234
>>> t[-2]
100
>>> t[-3]
'huevos'
>>> t[-4]
'pan'
>>> t[1:3]
('huevos', 100)
>>> t[0]
'pan'
```

Iteración sobre tuplas

Análogamente a lo que vimos con Listas, podemos iterar sobre los valores de una tupla.

```
>>> def imprimeValoresTupla(tupla):
        for elemento in tupla:
            print(elemento)

>>> imprimeValoresTupla((2,4,6))
2
4
6
>>> imprimeValoresTupla(())
>>> l = (99, True, 'una tupla',[1,2,3,4])
>>> imprimeValoresTupla(l)
99
True
una tupla
[1, 2, 3, 4]
```

Iteración sobre tuplas

También podemos escribir una versión recursiva de la función:

```
>>> def imprimeValoresTupla(tupla):
        if tupla != ():
                print(tupla[0])
                imprimeValoresTupla(tupla[1:])
```

Generando la misma salida anterior:

```
>>> imprimeValoresTupla((2,4,6))
2
4
6
>>> imprimeValoresTupla(())
>>> imprimeValoresTupla(1)
99
True
una tupla
[1, 2, 3, 4]
```

Tuplas

Entonces, ¿cuál es la diferencia con las Listas? Las tuplas no poseen los mecanismos de modificación, inserción y eliminación que mencionamos anteriormente, es decir:

```
>>> t
('pan', 'huevos', 100, 1234)
>>> t[0] = 1
Traceback (most recent call last):
  File "<pyshell#34>", line 1, in <module>
    t[0] = 1
TypeError: 'tuple' object does not support item assignment
```

Además, las tuplas son **inmutables**, es decir, sus valores no se pueden modificar una vez creada; y tienen un tamaño fijo.

Entonces, ¿para qué nos sirven las tuplas? Analicemos esto con un ejercicio:

Queremos almacenar la información: nombre, apellido y teléfono, de los alumnos de un curso.

Para esto, vamos a proponer una representación para cada alumno de la forma:

```
(nombre, apellido, teléfono)
```

La pregunta que a uno le puede surgir es por qué no usar listas en lugar de tuplas. La respuesta es sencilla, a la lista le puedo insertar datos, modificando la estructura que queremos que tenga cada alumno.

Tuplas

Las listas van a formar parte de la solución. ¿De qué manera? Como no sabemos, de antemano, cuantos alumnos vamos a tener, necesitamos una lista para guardar la información de los alumnos.

Entonces, vamos a tener una lista de tuplas, de la forma:

```
[(nombre1,apellido1,telefono1), (nombre2,apellido2,telefono2), ...]
```

Escribamos un par de funciones para ir armando nuestro programa.

Lo que necesitamos, ante todo, es una función que dada una lista de alumnos permita agregar un nuevo alumno.

Hay varias formas de hacer esto, vamos a proponer una:

```
>>> def agregaAlumno(listaDeAlumnos):
        print('Se va a agregar un alumno a la lista de alumnos existentes')
        nombre = input('Nombre: ')
        apellido = input('Apellido: ')
        telefono = input('Telefono: ')
        listaDeAlumnos += [(nombre, apellido, telefono)]
```

Notemos la forma en que armamos la tupla, donde usamos las variables en la posición que queremos los datos. También podemos mencionar que el teléfono es almacenado como String y no como entero para permitir algún caracter especial.

Veamos la función en uso:

```
>>> listaDeAlumnos = []
>>> agregaAlumno(listaDeAlumnos)
Se va a agregar un alumno a la lista de alumnos existentes
Nombre: Juan
Apellido: Perez
Telefono: 3415999999
>>> listaDeAlumnos
[('Juan', 'Perez', '3415999999')]
>>> agregaAlumno(listaDeAlumnos)
Se va a agregar un alumno a la lista de alumnos existentes
Nombre: Ana
Apellido: Perez
Telefono: 34158888888
>>> listaDeAlumnos
[('Juan', 'Perez', '3415999999'), ('Ana', 'Perez', '34158888888')]
```

Tuplas

Ahora que ya podemos agregar alumnos, vamos a escribir una función que tome una lista de alumnos y nos genere una salida amigable por pantalla:

```
>>> def muestraListaAlumnos(listaDeAlumnos):
        for (nombre,apellido,telefono) in listaDeAlumnos:
            print('El alumno', nombre, apellido, 'tiene el telefono:', telefono)
```

Analicemos la forma que tiene este for:

```
for (nombre,apellido,telefono) in listaDeAlumnos:
        print('El alumno', nombre, apellido, 'tiene el telefono:', telefono)
```

Lo que estamos haciendo es iterando sobre la lista usando que cada elemento de la misma va a ser una tupla formada por 3 datos. En este caso, nuestra forma de iterar es directamente con una tupla cuyo primer dato lo llamamos nombre, al segundo apellido y, al tercero teléfono.

Esta forma de escribirlo es similar a haber hecho esto:

```
for alumno in listaDeAlumnos:
        print('El alumno', alumno[0], alumno[1], 'tiene el telefono:', alumno[2])
```

Donde acá estamos llamando **alumno** al dato que hay en la lista y luego, accedo a las componentes de la tupla al mostrar los datos.

En ambos casos la salida es:

```
>>> muestraListaAlumnos(listaDeAlumnos)
El alumno Juan Perez tiene el telefono: 3415999999
El alumno Ana Perez tiene el telefono: 34158888888
```