Python

Estructuras tipos

Tupla y Diccionario

PROGRAMACION 2

Esc. Sup. De Comercio N° 49 Cap. Gral. J.J. de Urquiza

Técnico Superior en Desarrollo de Software

Tuplas

- Las tuplas son listas inmutables, es decir, no se pueden modificar después de su creación.
  - No permiten añadir, eliminar, mover elementos (no append, extend, remove)
  - Sí permiten extraer porciones, pero el resultado de la extracción es una tupla nueva
  - No permiten búsquedas (no index)
  - Sí permiten comprobar si un elemento se encuentra en la tupla.
- ¿Qué utilidad o ventaja tienen respecto a las listas?
  - Más rápidas
  - Menos espacio (mayor optimización)
  - Formatean Strings
  - Pueden utilizarse como claves en un diccionario (las listas no)

La sintaxis es nombre de la tupla y los elementos entre paréntesis **()**, los paréntesis son opcionales aunque se recomienda ponerlos por una cuestión de claridad del código.

```
tuplas=("lunes", "martes", "miercoles", "Jueves")
```

➔ Para hacer referencia a un elemento el procedimiento es igual que en las listas.

`print(tuplas[1])` mostrará martes

Conversión de tuplas a listas y viceversa.

Podemos convertir listas en tuplas y viceversa **list tuple**

```
listas=list(tuplas)
tuplas=tuple(listas)
```

- Contar cuantas veces está un elemento dentro de la tupla **count**

  `print(tuplas.count("lunes")`

- Cantidad (longitud) de elementos también utilizamos **len**

- Para saber si un elemento está en la tupla usamos **in**

  `print("lunes" in tuplas)`

  mostrará True o False de acuerdo si está o no

- Para crear una tupla unitaria, es decir con un único elemento debemos tener presente lo siguiente:

  `tuplas=("lunes",)` si no está la coma no creará una tupla de un elemento.

Empaquetado y desempaquetado de tuplas.

Podemos generar una tupla asignando a una variable un conjunto de variables o valores separados por coma, a esto se llama empaquetado de tupla

```
tupla="Juan","Perez",31482709
```

Puede asignar el contenido de la tupla a diferentes variables realizando lo siguiente:

```
tupla=(“Juan”,”Perez”,31482709)
nombre, apellido, dni=tupla
```

asignará automáticamente los elementos de la tupla a las variables siguiendo el orden, a esto se llama desempaquetado de tuplas

Ejemplo

Definir una tupla con tres valores enteros. Convertir el contenido de la tupla a tipo lista. Modificar la lista y luego convertir la lista en tupla.

```
fechatupla1=(25, 12, 2016)
print("Imprimimos la primer tupla")
print(fechatupla1)
fechalista=list(fechatupla1)
print("Imprimimos la lista que se le copio la tupla anterior")
print(fechalista)
fechalista[0]=31
print("Imprimimos la lista ya modificada")
print(fechalista)
fechatupla2=tuple(fechalista)
print("Imprimimos la segunda tupla que se le copio la lista")
print(fechatupla2)
```

Listas y tuplas anidadas

La lista es una estructura mutable (es decir podemos modificar sus elementos, agregar y borrar) en cambio una tupla es una secuencia de datos inmutable, es decir una vez definida no puede cambiar.

En Python vimos que podemos definir elementos de una lista que sean de tipo lista, en ese caso decimos que tenemos una lista anidada.

Ahora que vimos tuplas también podemos crear tuplas anidadas.

En general podemos crear y combinar tuplas con elementos de tipo lista y viceversa, es decir listas con componente tipo tupla.

Por ejemplo definimos la lista llamada empleado con tres elementos: en el primero almacenamos su nombre, en el segundo su edad y en el tercero la fecha de ingreso a trabajar en la empresa (esta se trata de una tupla) Podemos más adelante durante la ejecución del programa agregar otro elemento a la lista con por ejemplo la fecha que se fue de la empresa:

```
empleado=["juan", 53, (25, 11, 1999)]
print(empleado)
empleado.append((1, 1, 2016))
print(empleado)
```

Tenemos definida la tupla llamada alumno con dos elementos, en el primero almacenamos su nombre y en el segundo una lista con las notas que ha obtenido hasta ahora:

```
alumno=("pedro",[7, 9])
print(alumno)
```

Podemos durante la ejecución del programa agregar una nueva nota a dicho alumno:

```
alumno[1].append(10)
print(alumno)
```

Diccionario

El diccionario, define una relación uno a uno entre claves y valores.

La estructura de datos tipo diccionario utiliza una **clave** para acceder a un valor. El subíndice puede ser un entero, un float, un string, una tupla etc. (en general cualquier tipo de dato inmutable).

Podemos relacionarlo con conceptos que conocemos:

- Un diccionario tradicional que conocemos podemos utilizar un diccionario de Python para representarlo. La clave sería la palabra y el valor sería la definición de dicha palabra.
- Una agenda personal también la podemos representar como un diccionario. La fecha sería la clave y las actividades de dicha fecha sería el valor.
- Un conjunto de usuarios de un sitio web podemos almacenarlo en un diccionario. El nombre de usuario sería la clave y como valor podríamos almacenar su mail, clave, fechas de login etc.

Hay muchos problemas de la realidad que se pueden representar mediante un diccionario de Python.

Recordemos que las listas son mutables y las tuplas inmutables. Un diccionario es una estructura de datos mutable es decir podemos agregar elementos, modificar y borrar.

Definición de un diccionario por asignación.

```
productos={"manzanas":39, "peras":32, "lechuga":17}
print(productos)
```

Como vemos debemos encerrar entre llaves **{}** los elementos separados por coma. A cada elementos debemos indicar del lado izquierdo del carácter**:** la clave y al lado derecho el valor asignado para dicha clave. Por ejemplo para la clave "peras" tenemos asociado el valor entero 32.

Ejemplos

```
diccionario = {'nombre' : 'Carlos', 'edad' : 22, 'cursos':
['Python','Django','JavaScript'] }
```

Podemos acceder al elemento de un Diccionario mediante la clave de este elemento, como veremos a continuación:

```
print diccionario['nombre'] #Carlos
print diccionario['edad']#22
print diccionario['cursos'] #['Python','Django','JavaScript']
```

También es posible insertar una lista dentro de un diccionario. Para acceder a cada uno de los cursos usamos los índices:

```
print diccionario['cursos'][0]#Python
   print diccionario['cursos'][1]#Django
   print diccionario['cursos'][2]#JavaScript
```

Para recorrer todo el Diccionario, podemos hacer uso de la estructura for:

```
   for key in diccionario:
     print key, ":", diccionario[key]
```

En el bloque principal del programa definir un diccionario que almacene los nombres de paises como clave y como valor la cantidad de habitantes. Implementar una función para mostrar cada clave y valor.

```
paises={"argentina":40000000, "españa":46000000, "brasil":190000000, "uruguay": 3400000}
for clave in paises:
        print(clave, paises[clave])
```

Crear un diccionario que permita almacenar 5 artículos, utilizar como clave el nombre de productos y como valor el precio del mismo. Imprimir solo los artículos con precio superior a 100.

```
productos={}
for x in range(5):
    nombre=input("Ingrese el nombre del producto:")
    precio=int(input("Ingrese el precio:"))
    productos[nombre]=precio

print("Listado de articulos con precios mayores a 100")
for nombre in productos:
    if productos[nombre]>100:
       print(nombre)
```

Operador in con diccionarios

Para consultar si una clave se encuentra en el diccionario podemos utilizar el operador in:

```
if clave in diccionario:
    print(diccionario[clave])
```

Esto muy conveniente hacerlo ya que si no existe la clave produce un error al tratar de accederlo:

```
print(diccionario[clave])
```

Diccionarios: con valores de tipo listas, tuplas y diccionarios

Lo más poderoso que podemos encontrar en las estructuras de datos en Python es que podemos definir elementos que sean también estructuras de datos. En general se dice que podemos anidar una estructura de datos dentro de otra estructura de datos.
Ya vimos en conceptos anteriores que podemos definir elementos de una lista que sean también de tipo lista o de tipo tupla.
Hemos dicho que un diccionario consta de claves y valores para esas claves. Desarrollaremos problemas donde los valores para esas claves sean tuplas y o listas.

Métodos de los Diccionarios

dict ()

Recibe como parámetro una representación de un diccionario y si es factible, devuelve un diccionario de datos.

```
dic =  dict(nombre='nestor', apellido='Plasencia', edad=22)
dic → {‘nombre’ : 'nestor', ‘apellido’ : 'Plasencia', ‘edad’ : 22}
```

zip()

Recibe como parámetro dos elementos iterables, ya sea una cadena, una lista o una tupla. Ambos parámetros deben tener el mismo número de elementos. Se devolverá un diccionario relacionando el elemento i-esimo de cada uno de los iterables.

```
dic = dict(zip('abcd',[1,2,3,4]))
dic →   {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
```

items()

Devuelve una lista de tuplas, cada tupla se compone de dos elementos: el primero será la clave y el segundo, su valor.

```
dic =   {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
items = dic.items()
items → [(‘a’,1),(‘b’,2),(‘c’,3),(‘d’,4)]
```

keys()

Retorna una lista de elementos, los cuales serán las claves de nuestro diccionario.

```
dic =  {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
keys= dic.keys()
keys→ [‘a’,’b’,’c’,’d’]
```

values()

Retorna una lista de elementos, que serán los valores de nuestro diccionario.

```
dic =  {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
values= dic.values()
values→ [1,2,3,4]
```

clear()

Elimina todos los ítems del diccionario dejándolo vacío.

```
dic 1 =  {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
dic1.clean()
dic1 → { }
```

copy()

Retorna una copia del diccionario original.

```
dic = {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
dic1 = dic.copy()
dic1 → {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
```

fromkeys()

Recibe como parámetros un iterable y un valor, devolviendo un diccionario que contiene como claves los elementos del iterable con el mismo valor ingresado. Si el valor no es ingresado, devolverá none para todas las claves.

```
dic = dict.fromkeys(['a','b','c','d'],1)
dic →  {‘a’ : 1, ’b’ : 1, ‘c’ : 1 , ‘d’ : 1 }
```

get()

Recibe como parámetro una clave, devuelve el valor de la clave. Si no lo encuentra, devuelve un objeto none.

```
dic = {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
valor = dic.get(‘b’)
valor → 2
```

pop()

Recibe como parámetro una clave, elimina esta y devuelve su valor. Si no lo encuentra, devuelve error.

```
dic = {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
valor = dic.pop(‘b’)
valor → 2
dic → {‘a’ : 1, ‘c’ : 3 , ‘d’ : 4}
```

setdefault()

Funciona de dos formas. En la primera como get

```
dic = {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
valor = dic.setdefault(‘a’)
valor → 1
```

Y en la segunda forma, nos sirve para agregar un nuevo elemento a nuestro diccionario.

```
dic = {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
valor = dic.setdefault(‘e’,5)
dic → {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4 , ‘e’ : 5}
```

update()

Recibe como parámetro otro diccionario. Si se tienen claves iguales, actualiza el valor de la clave repetida; si no hay claves iguales, este par clave-valor es agregado al diccionario.

```
dic 1 = {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
dic 2 = {‘c’ : 6, ’b’ : 5, ‘e’ : 9 , ‘f’ : 10}
dic1.update(dic 2)
dic 1 → {‘a’ : 1, ’b’ : 5, ‘c’ : 6 , ‘d’ : 4 , ‘e’ : 9 , ‘f’ : 10}
```