Clase 10 Tuplas/Conjuntos/Diccionarios

Concepto, Operaciones y Aplicaciones

Teoria Tuplas

- Las tuplas son similares a las listas , ya que se accede a ellas a traves de un subindice.
- Para crear una tupla se encierran sus elementos entre paréntesis en lugar de corchetes:

```
salsas = ("tartara","criolla","chimichurri")
```

- Si se escribe una secuencia de elementos sin paréntesis pero separados por comas, también crea una tupla.

```
salsas = "tartara" , "criolla" , "chimichurri"
```

- Una tupla vacía se define con un par de paréntesis , sin nada en su interior.

```
aderezos = ()
```

- Para crear una tupla con un solo elemento es necesario escribir una coma luego de este.

```
misalsa = "barbacoa", #si no hay com es cadena
```

Diferencia con las listas: **Las tuplas son inmutables** , no pueden ser modificadas

- Como son inmutables carecen del método append().
- La unica forma de agregar elementos es a traves del operador de concatenación creando una ***nueva tupla***:

```
aderezos = aderezos + ("mayones",)
```

- El operador de repetición * también puede ser usado con tuplas.

```
binario = (0,1) *3
```

```
print(binario) #(0,1,0,1,0,1)
```

- En una misma tupla se pueden combinar distintos tipos de datos. Y también pueden ser anidadas.
- tuplas → datos hetereogeneos
- strings → datos homogéneos

```
primavera = (21,"Septiembre")
#anidadas
alumno = (152308,"Luis Arce","Lima 7
17",(1,"Mayo",1998))
```

- Al igual que las listas se pueden acceder a sus elementos mediante un subíndice (base 0), o mas de uno si las tuplas están anidadas:

```
alumno = (152308,"Luis Arce","Lima 7
17",(1,"Mayo",1998))
print(alumno[1]) #Luis Arce
print(alumno[3][1]) #Mayo
```

- Tratar de modificar un elemento de una tupla mediante su subíndice provocara una excepción de tipo **TypeError**.

```
t = (1,2,3)
t[0] = 4
TypeError: 'tuple' object does not s
upport item assignment
```

- Como todo iterable también pueden ser recorridas por un for. Y manipuladas mediante rebanadas.

```
for dato in alumno:
    print(dato)
#Rebanadas
print(alumno[ :2] #(15314,"Luis Arc
e")
```

Funciones y metodos

- Las funciones **len() max() min() sum()** operan con tuplas igual que lo hacen con listas.
- Tambien actúan del mismo modo el operador in y los métodos index y count.
- Sin embargo, no están disponibles los metodos que rabajan in situ, como **append** , **remove** , **pop** o **sort**

Conversion de tuplas

- Una lista puede ser convertida en tuplas mediante la función tuple().
- Una tupla puede ser convertida en lista con la función list().

```
lista = [1,2,3]
tupla = tuple(lista) #(1,2,3)
```

Aplicaciones

- Mediante el uso de tuplas es posible intercambiar el valor de dos variables sin necesidad de usar una variable auxiliar:
  `a , b = b , a`

```
fecha = (dia,mes,año)
# cad nro es diferente, no sumamos n
ada
```

👉 **Por lo general se prefiere utilizar tuplas cuando los elementos son heterogéneos, y listas cuando estos son homogéneos.**

Teoria Conjuntos

- Un *conjunto* es una colección de elementos sin orden ni duplicados.
- No entran dentro de la categoría *secuencia* porque carecen de orden interno.
- Se los suele utilizar cuando se desea eliminar elementos repetidos.
- Para crear un conjunto se procede en forma similar a las listas, pero reemplazando los corchetes por llaves:

```
frutas = {"banana","manzana","naranja","pera","banana"}
print(frutas)
#{"banana","manzana","naranja","pera"}
                    #banana solo quedo una vez
```

- Un conjunto vacío se crea utilizando la función **set( )**:

`conjunto = set( )`

- Si se escriben dos llaves juntas se crea un diccionario.
- Los conjuntos son ***mutables***, se pueden modificar.
- Sin embargo, los elementos que se agreguen a un conjunto deben ser de un tipo ***inmutable*** (numeros, string o tuplas).
- Tratar de agregar un dato perteneciente a un tipo mutable (listas, diccionarios u otros conjuntos) provocara un error:
- >>>lenguajes = {[”Python”,Pearl”],[”Java”,”C++”,”C#”]} ***TypeError: unhashable type: ‘list’***

Operaciones con conjuntos

- Para verificar si un elemento se encuentra dentro de un conjunto se utiliza el operador **in** (o **not in**):

```
if "manzana" in frutas:
```

```
    print("Verde o roja?")
```

- Las operaciones habituales de conjuntos se realizan con los siguientes operadores:
  - **Union `|` no hay repetidos**
  - **Diferencia `-`**
  - **Intersección `&`**
  - **Diferencia Simétrica `^`**

Funciones

- Las funciones `len()` `max()` `min()` `sum()` tambien operan con conjuntos.
- Por tratarse de **iterables**, los conjuntos pueden ser recorridos mediante un ciclo for:
- no hay orden. no hay garantías conrespecto a como lo va a devolver python

```
conj = set(range(10))
for elem in conj:
    print(elem,end=" ")
```

Métodos

- El método `add(<elem>)` agrega un elemento al conjunto:

```
conjunto = {3,4,5}
conjunto.add(6) #Equivale al append en listas
print(conjunto) #{3,4,5,6}
```

- El método `remove(<elem>)` elimina un elemento identificado por su valor. Provoca una excepción *KeyError* si no esta presente.

```
conjunto = {3,4,5}
conjunto.remove(4)
print(conjunto) #{3,5}
```

- El método `discard(<elem>)` también elimina un elemento identificado por su valor. ***Este método no provoca una excepción si no se encuentra.***

```
conjunto = {3,4,5}
conjunto.discard(4)
print(conjunto) #{3,5}
```

- El método `pop()` elimina y devuelve un elemento del conjunto ***elegido al azar***:
- Provoca un ***KeyError*** si el conjunto esta vacío.

```
conjunto = {3,4,5}
x = conjunto.pop()
print(conjunto) #{?,?}
```

- El metodo `clear()` elimina todos los elementos del conjunto.

```
conjunto = {3,4,5}
conjunto.clear()
```

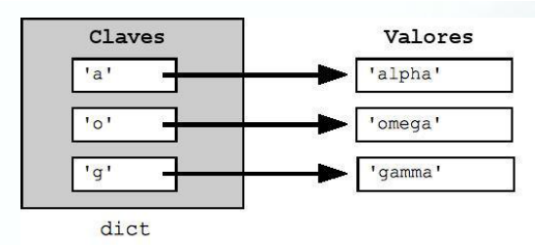
```
print(conjunto) #set()
```

EJEMPLO 1

Desarrollar un programa para simular la entrega de naipes a un jugador de poker , evitando la generación de naipes repetidos.

```
import random
simbolo = ("Trebol","Pica","Corazon","Diamante") #los cuatro palos. tupla
mano = set()
intentos = 0
while len(mano)<5: #Cinco cartas por jugador
    numero = random.randint(1,13) #1 a 10 mas J,Q,K
#     palo = random.randint(0,3) #subíndice ---> simbolo[palo]
    carta = (numero,random.choice(simbolo)) #tupla
    mano.add(carta)
    intentos = intentos + 1
print(mano)
print("Intentos realizados: ",intentos)
```

Teoria Diccionarios



- Son estructuras de datos que se utilizan para relacionar claves y valores.
- Cada elemento de un diccionario se representa mediante una dupla `clave:valor`
- Se crean encerrando sus duplas entre llaves y separándolas por comas.

```
edades = {"Dante":27,"Brenda":18,"Ma
lena":23}
```

- Para acceder a sus elementos se utiliza la clave en lugar de un subíndice numérico.

```
edades = {"Dante":27,"Brenda":18,"Ma
lena":23}
print(edades["Brenda"]) #18
```

- La clave de cada dupla debe pertenecer a un tipo **inmutable** (números , cadena de caracteres , tuplas).

```
'''ejemplo con colores rgb'''
colores = {"Rojo":[255,0,0],"Verde":
```

- Los valores asociados a cada clave pueden ser de **cualquier tipo**, incluyendo listas u otros diccionarios.

```
[0,255,0],"Azul":[0,0,255]}
```

- Los diccionarios no son secuencias, por lo tanto no están ordenados.
- No se puede utilizar un subíndice para acceder a sus elementos.
- Funcionan como una lista a la que se accede mediante una clave.

- Los diccionarios pueden definirse con un formato mas claro y legible, colocando cada dupla debajo de la anterior.

```
colores = {
    "Rojo" : [255,0,0],
    "Verde": [0,255,0],
    "Azul" : [0,0,255]
}
```

- Al igual que ocurre con las listas, también puede crearse diccionarios por comprensión. `listas por comprensión`

```
dic = {x:x**2 for x in range(1,5)}
print(dic) #{1:1 , 2:4, 3:9, 4:16}
```

- Las rebanadas no son aplicables a los diccionarios ya que carecen de orden interno. Las claves deben ser únicas; no se permiten claves duplicadas.

- Asignar un valor a una clave reemplaza el valor existente o crea una nueva clave nueva , dependiendo de si existía o no.

```
Colores["Gris"] = [128,128,128]
```

- Para recorrer un diccionario es posible utilizar la instrucción `for`. La variable del for recibe el valor de cada clave del diccionario.

```
for color in colores:
    print(color, "" , colores[colo
r])
```

aclaraciones_____

- No es posible acceder a una clave a traves de su valor.
- Un mismo valor puede estar asociado a mas de una clave.
- Tratar de acceder a un elemento con una clave inexistente provoca una excepción `KeyError`

- Puede verificarse si una clave existe utilizando el operador **in** (o **not in**).
- Tambien es posible utilizar el método `get()` , que devuelve el valor asociado a una clave o None si la misma no se encuentra. Tambien acepta un segundo parámetro que sera

```
a = colores["Rojo"]
b = colores.get("Rojo")

a = colores.get("Cian","No encontrad
o")
```

devuelto en lugar de None cuando la clave no este presente.

- El método `pop(<clave> [,<valor>])` busca `<clave>` dentro del diccionario y la elimina, devolviendo el valor asociado a esa clave. Si la clave no se encuentra devuelve `<valor>`. Si <valor> no se incluye se produce un `KeyError`.

```
a = colores.pop("Verde")
print(a) #[0,255,0]
```

- El método `keys()` devuelve una secuencia con las claves presentes en el diccionario.
- La funcion `list()` es necesario para convertir el objeto devuelto por el metodo

```
clave = list(colores.keys())
print(clave) #['Rojo','Verde','Azu
l']
```

- El metodo `values()` devuelve una secuencia con los valores presentes en el diccionario.

```
valores = list(colores.values())
print(valores) #[[255,0,0],[0,255,
0],[0,0,255]]
'''list() cumple la misma tarea ante
rior'''
```

- El metodo `items()` devuelve una secuencia de tuplas `clave:valor`
- devuelve una tupla

```
for color, RGB in colores.items():
    print(color,"->",RGB)
'''La secuencia puede ser convertida
a lista mediante la funcion list
().'''
```

Método *fromkeys*

- `fromkeys(<secuencia>[,<valor>])` sirve para crear un diccionario a partir de una secuencia cualquiera (lista,cadena,tuplas,rango...).
- Cada elemento de la secuencia se convierte en una clave del diccionario.
- El valor asociado sera *None* u otro proporcionado por el programador.
- Este metodo no se aplica sobre una variable sino sobre la clase `dict`.
- Devuelve el diccionario creador como valor de retorno.

```
dias = "Lunes","Martes" #Tupla
d1 = dict.fromkeys(dias)
print(d1) #{'Lunes':None,'Martes':None}
```

```
vocales = "aeiou" #String
d2 = dict.fromkeys(vocales,0)
print(d2) #{'a':0 , 'e':0, 'i':0, 'o':0, 'u':0}
```

Instrucción del

- Ademas del metodo pop,la instruccion del se utiliza para eliminar un elemento de un diccionario. `del colores[”Rojo”]` . Tambien permite borraar el diccionario completo `del colores`

> Ejemplo 2: Realizar un programa para ingresar una frase y mostrar un listado ordenado alfabeticamente con las palabras que contiene, eliminando las repetidas y añadiendo junto a cada una la cantidad de veces que se encontro.

```
frase = int(input("Ingrese una frase:\n")
listadepalabras = frase.split()
dic = {}
for palabra in listadepalabras:
    if palabra not in dic:
        dic[palabra] = 1
    else:
        dic[palabra] = dic[palabra] + 1
listado = []
for p in dic:
    listado.append(">"+p":"+str(dic[p])+"veces")
listado.sort()
for linea in listado:
    print(linea)
```

Aplicaciones

Algunas aplicaciones de los diccionarios son:

- Agenda telefonica
- Control de stock
- Usuario y contraseñas
- Listas de precios