Clase Nº 10

Tuplas, Conjuntos y Diccionarios

Tuplas

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

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

Tuplas

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

```
salsas = "tártara", "criolla", "chimichurri"
```


Tuplas

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

```
aderezos = ( )
```

Tuplas

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

```
misalsa = "barbacoa",
```

- Los paréntesis también son opcionales en este caso.

Tuplas

- Esta propiedad puede utilizarse para concatenar un nuevo elemento mediante el operador +, aunque en este caso los paréntesis son obligatorios:

```
salsas = salsas + ("provenzal",)
```

Tuplas

- La principal diferencia entre listas y tuplas es que las tuplas ***son inmutables***, y por lo tanto no pueden ser modificadas.
- Tratar de alterar una tupla provoca una excepción de tipo TypeError.

Tuplas

- Debido a que son inmutables, las tuplas carecen del método *append( )*.
- La única forma de agregar elementos es a través del operador de concatenación, creando una nueva tupla:

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

Tuplas

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

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


Tuplas

- En una misma tupla se pueden combinar distintos tipos de datos.

```
primavera = (21, "Septiembre")
```

- y también pueden ser anidadas.

```
alumno = (152408, "Luis Arce", "Lima 717", (1, "Mayo", 1998))
```

Tuplas

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

```
alumno = (152408, "Luis Arce", "Lima 717", (1, "Mayo", 1998))
print(alumno[1])      # Luis Arce
print(alumno[3][1])   # Mayo
```


Tuplas

- Como todo iterable, pueden ser recorridas mediante un ciclo *for*:

```
for dato in alumno:
    print(dato)
```

- y también pueden ser manipuladas mediante rebanadas:

```
print(alumno[:2])  # (152408,"Luis Arce")
```

Tuplas

- Utilizando la técnica de las rebanadas es posible crear una nueva tupla a partir de otra.

```
tupla1 = (1, 2, 4, 5)
tupla2 = tupla1[:2] + (3,) + tupla1[2:]
print(tupla2)  # (1, 2, 3, 4, 5)
```


Funciones y métodos

- Las funciones **len()**, **max()**, **min()** y **sum()** operan con tuplas igual que lo hacen con listas.
- También actúan del mismo modo el operador **in** y los métodos **index** y **count**.
- Sin embargo, no están disponibles los métodos **append**, **remove** o **pop** debido a que las tuplas son ***inmutables***.

Conversión de tuplas

- Una lista puede ser convertida en tupla 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)
```


Empaquetado

- El ***empaquetado de tuplas*** es el proceso a través del cual una serie de valores simples se convierten en una tupla:

```
dia = 1
mes = "Mayo"
año = 2018
fecha = dia, mes, año
```

Desempaquetado

- El *desempaquetado de tuplas* es el proceso inverso del anterior, donde una tupla de longitud N se asigna a un conjunto de N variables simples:

dia, mes, año = fecha

Aplicaciones

- Las tuplas son más rápidas que las listas al momento de ser recorridas, pero es necesario recordar que no se pueden modificar.

Aplicaciones

- Por lo general se prefiere utilizar tuplas cuando los elementos son heterogéneos, y listas cuando éstos son homogéneos.

  fecha = (dia, mes, año)

- Aunque todos son números, su significado es heterogéneo.

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 para verificar la presencia de elementos y cuando se desea eliminar repetidos.

Conjuntos

- **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" quedó sólo una vez
```


Conjuntos

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

```
conjunto = set( )
```

- **Si se escriben dos llaves juntas se crea un *diccionario*.**

Conjuntos

- Los conjuntos son **mutables**, es decir que *pueden ser modificados*.
- Sin embargo, los elementos que se agreguen a un conjunto deben ser de un tipo ***inmutable***.

Conjuntos

- Tratar de agregar un dato perteneciente a un tipo mutable (listas, diccionarios u otros conjuntos) provocará un error:

```
>>> lenguajes = {["Python","Perl"], ["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?")
```


Operaciones con conjuntos

- Las operaciones habituales de conjuntos se realizan con los siguientes operadores:
  - | unión A U B
  - & intersección A ∩ B
  - – diferencia A – B
  - ^ diferencia simétrica A ∆ B

Operaciones con conjuntos

```
primos = {1, 2, 3, 5, 7}
pares = {2, 4, 6, 8}
impares = {1, 3, 5, 7, 9}
numeros = pares | impares  # Unión
                           # {1, 2, 3, 4, 5, 6, 7, 8, 9}
impprimos = impares & primos    # Intersección
                                # {1, 3, 5, 7}
```


Operaciones con conjuntos

```
primos = {1, 2, 3, 5, 7}
impares = {1, 3, 5, 7, 9}
numeros = pares | impares
noprimos = numeros – primos     # Diferencia
                                # {4, 6, 8, 9}

pares = numeros ^ impares       # Dif. Simétrica
                                # {2, 4, 6, 8}
```

Funciones

- Las funciones len( ), max( ), min( ) y sum( ) operan con conjuntos igual que lo hacen con listas y tuplas.
- También pueden ser recorridos mediante un ciclo for:

```
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)
print(conjunto)    # {3, 4, 5, 6}
```

- Equivale al *append* de las listas.

Métodos

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

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


Métodos

- El método discard(<elem>) también elimina un elemento identificado por su valor. Es similar al anterior, pero no provoca una excepción si el elemento no se encuentra.

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

Métodos

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

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


Métodos

- El método issubset(<conj>) devuelve *True* si el conjunto está incluido dentro de <conj>.

```
conjunto = {3, 4, 5}
if conjunto.issubset({2, 3, 4, 5, 6}):
    print("Incluido")  # "Incluido"
```

Ejemplo 1

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

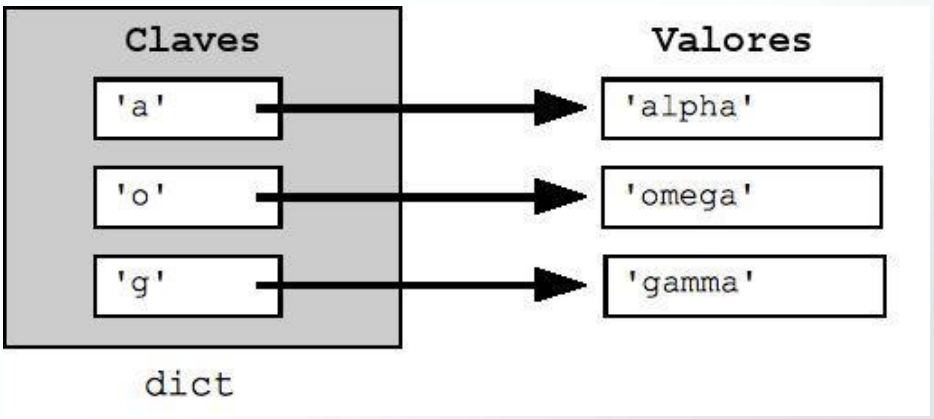

```
import random

simbolo = ("Trébol", "Pica", "Corazón", "Diamante")
mano = set( )
intentos = 0
while len(mano)<5:  # cinco cartas por jugador
    numero = random.randint(1,13)  # 1 a 10 más J, Q, K
    palo = random.randint(0,3)
    carta = (numero, simbolo[palo])
    mano.add(carta)
    intentos = intentos + 1
print(mano)
print("Intentos realizados:", intentos)
```

Diccionarios

- **Son estructuras de datos que se utilizan para relacionar claves y valores.**
- **También se los conoce como *arreglos asociativos* o *tablas de hash*.**

Diccionarios



Diccionarios

- 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, "Malena":23}
```


Diccionarios

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

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

Diccionarios

- La clave de cada dupla debe pertenecer a un tipo ***inmutable*** (números, cadenas de caracteres, tuplas).
- Los valores asociados a cada clave pueden ser de ***cualquier*** tipo, incluyendo listas u otros diccionarios.

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


Diccionarios

- Los diccionarios no son secuencias, y 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.

Diccionarios

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

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


Diccionarios

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

Diccionarios

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

```
colores["gris"] = [128,128,128]
```


Diccionarios

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

Diccionarios

- 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.

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


Diccionarios

- El método get( ) admite un segundo parámetro que será devuelto en lugar de *None* cuando la clave no esté presente.

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

Diccionarios

- Para eliminar un elemento de un diccionario se utiliza la instrucción **del**:

```
del colores["Rojo"]
```

- ...que también permite eliminar el diccionario completo.

```
del colores
```


Diccionarios

- 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[color])
```

Diccionarios

- El método **keys( )** devuelve una lista con las claves presentes en el diccionario
- El método **items( )** devuelve una lista de tuplas ***clave:valor***.

```
for color, RGB in colores.items( ):
    print(color, "➔", RGB)
```


Diccionarios

- **fromkeys(<secuencia>[, <valor>])** sirve para crear un diccionario a partir de una secuencia cualquiera (lista, cadena, tupla…).
- Cada elemento de la secuencia se convierte en una clave del diccionario.
- El valor asociado será *None* u otro proporcionado por el programador.

Diccionarios

- Este método no se aplica sobre una variable sino sobre la clase ***dict***.
- Devuelve el diccionario creado como valor de retorno.

Diccionarios

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

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

Ejemplo 2

Realizar un programa para ingresar una frase y mostrar un listado ordenado alfabéticamente con las palabras que contiene, eliminando las repetidas y añadiendo junto a cada una la cantidad de veces que se encontró.


```
frase = 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)
```

Ejemplo de ejecución:

Ingrese una frase:

buenos son los viejos amigos, pero solo los buenos llegan a viejos

> a: 1 veces
> amigos,: 1 veces
> buenos: 2 veces
> llegan: 1 veces
> los: 2 veces
> pero: 1 veces
> solo: 1 veces
> son: 1 veces
> viejos: 2 veces

Aplicaciones

Algunas aplicaciones de los diccionarios son:

- **Agenda telefónica**
- **Control de stock**
- **Usuarios y contraseñas**
- **Listas de precios**

Ejercitación

- **Práctica 8: Completa**