Clase02_1_Tuplas_diccionarios

March 20, 2024

1 Seminario de Lenguajes - Python

1.1 Cursada 2024

1.2 Tuplas y diccionarios

Trabajamos con listas

¿Se acuerdan de este método de str?

[ ]:
```
my_list = "Somos campeones del  mundo!!!".split()
my_list
```

El método split, retorna una **lista** con los elementos de la cadena de caracteres.

Dijimos que una lista es una colección heterogénea de datos. Esto significa que:

[ ]:
```
my_list.append(10)
my_list
```

1.3 ¿Qué otras operaciones podemos hacer con listas?

2 Slicing con listas

- Al igual que en el caso de las cadenas de caracteres, se puede obtener una porción de una lista usando el operador **":"**

[ ]:
```
vowels = [ "a", "e","i", "o", "u"]
print(vowels[1:3])
```

- Si no se pone inicio o fin, se toma por defecto las posiciones de inicio y fin de la lista.
- **Tarea para el hogar:** probar con índices negativos.

3 Tuplas: otro tipo de secuencias en Python

- Al igual que las listas, son colecciones heterogéneas de datos ordenados.

[ ]:
```
tuple = 1, 2
tuple1 = (1, 2)
tuple2 = (1,)  # OJO con esto
```

```
tuple3 = ()
type(tuple2)
```

Observemos detallamente el código del DESAFÍO 4 de la clase

[ ]:
```
words = ["casa", "ir", "sol", "cantar","correr"]
verbs = [pal for pal in words if pal.endswith(("ar", "er", "ir"))]
verbs
```

En este caso, estamos pasando una tupla como argumento al método **endswith**.

3.1 ¿Cuál es la diferencia con las listas?

Veamos las siguientes situaciones.

4 Tuplas vs. listas

[ ]:
```
my_tuple = (1, 2)
my_list = [1, 2]

elem = my_tuple[0]
elem
#print(len(my_tuple))
```

- Se acceden a los elementos de igual manera: usando [] (empezando desde cero)
- La función **len** retorna la cantidad de elementos en ambos casos.

¿Entonces?

4.1 DIFERENCIA: las tuplas son INMUTABLES

- Su tamaño y los valores de las mismas NO pueden cambiar.

[ ]:
```
my_tuple = (1, 2)
my_list = [1, 2]

my_tuple[0] = "uno" # Esto da error
my_tuple
#my_tuple.append("algo") # Esto da error
```

```
TypeError: 'tuple' object does not support item assignment
```

4.1.1 Tenemos que acostumbrarnos a leer los errores.

5 Obteniendo subtuplas

[ ]:
```
my_tuple = (1, 2, 3, "hola")
print(my_tuple[1:4])
new_tuple = ("nueva",) + my_tuple[:2]
print(new_tuple)
```

[ ]:
```
# ¿por qué da error este código?
new_tupla = ('nueva') + my_tuple[1:3]
new_tupla
```

6 DESAFÍO 1

Volvemos a procesar las películas de Dragon Ball.

Queremos saber: - cuál fue la duración, en minutos, promedio; y - **qué** películas duran más que el promedio, en minutos.

¿Qué diferencia hay con el desafío 1?

- Deberíamos ingresar no sólo las duraciones, sino también los nombres de las películas.
- ¿Qué soluciones proponen?

7 ¿Qué les parece esta solución?

[ ]:
```
movie = input("Ingresa el nombre de una película de Dragon Ball (<FIN> para␣
 ↪finalizar)")
movies_list = []
while movie != "FIN":
    duration = int(input(f"Ingresa la duración de la película {movie}"))
    movies_list.append((movie, duration))
    movie = input("Ingresa el nombre de una película de Dragon Ball (<FIN> para␣
 ↪finalizar)")
movies_list
```

- ¿Qué estructura de datos estoy usando?
- Hay algo mejor…

8 Diccionarios en Python

- Un diccionario es un conjunto **no ordenado** de pares de datos: **clave:valor**.
- Se definen con **{ }**.

[ ]:
```
movies = {"Dragon Ball: La leyenda de Shenron": 50,
          "Dragon Ball Z: Los guerreros de plata": 48,
```

```
    "Dragon Ball Z: La batalla de los dioses": 85,
    "Dragon Ball Z: La resurrección de Freezer": 93,
    "Dragon Ball Super: Broly": 100
}
```

[ ]:
```
movies["Dragon Ball: La leyenda de Shenron"]
```

9 Las claves deben ser únicas e inmutables

- Las **claves** pueden ser cualquier tipo **inmutable**.
  - Las cadenas y números siempre pueden ser claves.
  - Las tuplas se pueden usar sólo si no tienen objetos mutables.

[ ]:
```
# Probar cuáles de las siguientes instrucciones dan error

#my_dic = {"uno":1}
#my_dic = {1: "uno"}
#my_dic = {[1,2]: "lista"}
#my_dic = {(1,2): "tupla"}
#my_dic = {([1],2): "tupla"}
#my_dic
```

10 ¿Cómo accedemos a los elementos?

- Al igual que las listas y tuplas, se accede usando [ ] pero en vez de un índice que representa la posición, **usamos la clave**.
- Es un error extraer un valor usando una clave no existente.

[ ]:
```
months = {"enero": 31, "febrero": 28, "marzo": 31}
months
```

11 ¿Cómo agregamos elementos?

- Si se usa una clave que ya está en uso para guardar un valor, el valor que estaba asociado con esa clave se pierde, si no está la clave, se agrega.

[ ]:
```
months["febrero"] = 29
months["abril"] = 30
months
```

12 Volviendo al desafío planteado ...

- Nos falta saber cómo definir un diccionario vacío para luego ir agregando los valores.

```
[ ]: movie = input("Ingresa el nombre de una película de Dragon Ball (<FIN> para␣
      ↪finalizar)")
     dic_movies = {}
     while movie != "FIN":
         duration = int(input(f"Ingresa la duración de la película  {movie}"))
         dic_movies[movie] = duration
         movie = input("Ingresa el nombre de una película de Dragon Ball (<FIN> para␣
      ↪finalizar)")
     dic_movies
```

13 ¿Cómo recorremos un diccionario?

```
[ ]: music = {"rock": ["Riff", "La Renga", "La Torre"],
              "blues": ["La Mississippi", "Memphis"]}

     for elem in music:
         print(elem)
```

¿A qué referencia la variable **elem**?

Si queremos mostrar los valores:

```
[ ]: music = {"rock": ["Riff", "La Renga", "La Torre"],
               "blues": ["La Mississippi", "Memphis"]}

     for elem in music:
         print(music[elem])
```

14 Existen algunos métodos útiles:

```
[ ]: music_keys = music.keys()
     music_values = music.values()
     music_items = music.items()
```

```
[ ]: for elem in music_values:
         print(elem)
```

```
[ ]: music_keys
```

```
[ ]: type(music_keys)
```

Probar: ¿cómo procesamos los datos obtenidos por el método items?

15 El operador in en diccionarios

¿Verifica en las claves o los valores?

[ ]:
```
music = {"rock": ["Riff", "La Renga", "La Torre"],
         "blues": ["La Mississippi", "Memphis"]}
"rock" in music
```

16 Observemos este código

[ ]:
```
months = {"enero": 31, "febrero": 28, "marzo": 31}
months1 = months
months2 = months.copy()
print(id(months))
print(id(months1))
print(id(months2))
```

¿Qué significa?

[ ]:
```
months1["abril"] = 30
months2["abril"] = 43
months
```

17 Más operaciones

Probar:

- **del**: permite borrar un par clave:valor
- **clear()**: permite borrar todo

[ ]:
```
# Probamos del y clear
```

18 Otra forma de crear diccionarios

- Podemos usar **dict()**.
- Se denomia "constructor" y crea un diccionario directamente desde una secuencia de pares clave-valor.

[ ]:
```
months = dict([("enero", 31), ("febrero", 28), ("marzo", 31)])
months
```

- ¿Qué tipos de datos se pueden usar para la secuencia?
- ¿Y para los pares clave-valor?

19 Por comprensión

[ ]:
```
dict ([(x, x**2) for x in (2, 4, 6)])
```

[ ]:
```
import string

ascii_numbers = dict([(n, ord(n)) for n in string.digits])
ascii_numbers
```