# Puntos Clave: Tuplas

1. Las **Tuplas** son colecciones de datos ordenadas e inmutables. Se puede pensar en ellas como listas inmutables. Se definen con paréntesis:

```
my_tuple = (1, 2, True, "una cadena", (3, 4), [5, 6], None)

print(my_tuple)
```

```
my_list = [1, 2, True, "una cadena", (3, 4), [5, 6], None]

print(my_list)
```

Cada elemento de la tupla puede ser de un tipo de dato diferente (por ejemplo, enteros, cadenas, boleanos, etc.). Las tuplas pueden contener otras tuplas o listas (y viceversa).

2. Se puede crear una tupla vacía de la siguiente manera:

```
empty_tuple = ()

print(type(empty_tuple))     # salida: <class 'tuple'>
```

3. La tupla de un solo elemento se define de la siguiente manera:

```
one_elem_tuple_1 = ("uno", )     # Paréntesis y una coma.

one_elem_tuple_2 = "uno",        # Sin paréntesis, solo la coma.
```

Si se elimina la coma, Python creará una **variable** no una tupla:

```
my_tuple_1 = 1,
```

```
print(type(my_tuple_1))     # salida: <class 'tuple'>
```

```
my_tuple_2 = 1              # Esto no es una tupla.
print(type(my_tuple_2))     # salida: <class 'int'>
```

4. Se pueden acceder los elementos de la tupla al indexarlos:

```
my_tuple = (1, 2.0, "cadena", [3, 4], (5, ), True)
print(my_tuple[3])     # salida: [3, 4]
```

5. Las tuplas son **immutable**, lo que significa que no se puede agregar, modificar, cambiar o quitar elementos. El siguiente fragmento de código provocará una excepción:

```
my_tuple = (1, 2.0, "cadena", [3, 4], (5, ), True)
my_tuple[2] = "guitarra"     # La excepción TypeError será lanzada.
```

Sin embargo, se puede eliminar la tupla completa:

```
my_tuple = 1, 2, 3,
del my_tuple
print(my_tuple)     # NameError: name 'my_tuple' is not defined
```

6. Puedes iterar a través de los elementos de una tupla con un bucle (Ejemplo 1), verificar si un elemento o no esta presente en la tupla (Ejemplo 2), emplear la función `len()` para verificar cuantos elementos existen en la tupla (Ejemplo 3), o incluso unir o multiplicar tuplas (Ejemplo 4):

```
# Ejemplo 1
tuple_1 = (1, 2, 3)
for elem in tuple_1:
    print(elem)

# Ejemplo 2
tuple_2 = (1, 2, 3, 4)
print(5 in tuple_2)
print(5 not in tuple_2)

# Ejemplo 3
tuple_3 = (1, 2, 3, 5)
print(len(tuple_3))

# Ejemplo 4
tuple_4 = tuple_1 + tuple_2
tuple_5 = tuple_3 * 2

print(tuple_4)
print(tuple_5)
```

**EXTRA**

También se puede crear una tupla utilizando la función integrada de Python `tuple()`. Esto es particularmente útil cuando se desea convertir un iterable (por ejemplo, una lista, rango, cadena, etcétera) en una tupla:

```
my_tuple = tuple((1, 2, "cadena"))

print(my_tuple)


my_list = [2, 4, 6]

print(my_list)     # salida: [2, 4, 6]

print(type(my_list))     # salida: <class 'list'>

tup = tuple(my_list)

print(tup)     # salida: (2, 4, 6)

print(type(tup))     # salida: <class 'tuple'>
```

De la misma manera, cuando se desea convertir un iterable en una lista, se puede emplear la función integrada de Python denominada `list()`:

```
tup = 1, 2, 3,

my_list = list(tup)

print(type(my_list))     # salida: <class 'list'>
```

## Puntos Clave: Diccionarios

1. Los diccionarios son *colecciones indexadas de datos, mutables y desordenadas. (*En Python 3.6x los diccionarios están ordenados de manera predeterminada.

Cada diccionario es un par de *clave : valor*. Se puede crear empleado la siguiente sintaxis:

```
my_dictionary = {
    key1: value1,
    key2: value2,
    key3: value3,
    }
```

2. Si se desea acceder a un elemento del diccionario, se puede hacer haciendo referencia a su clave colocándola dentro de corchetes (Ejemplo 1) o utilizando el método `get()` (Ejemplo 2):

```
pol_esp_dictionary = {
    "kwiat": "flor",
    "woda": "agua",
    "gleba": "tierra"
    }

item_1 = pol_esp_dictionary["gleba"]     # Ejemplo 1.
print(item_1)     # salida: tierra

item_2 = pol_esp_dictionary.get("woda")     # Ejemplo 2.
print(item_2)     # salida: agua
```

3. Si se desea cambiar el valor asociado a una clave específica, se puede hacer haciendo referencia a la clave del elemento, a continuación se muestra un ejemplo:

```
pol_esp_dictionary = {
    "zamek" : "castillo",
    "woda"  : "agua",
    "gleba" : "tierra"
    }

pol_esp_dictionary["zamek"] = "cerradura"
item = pol_esp_dictionary["zamek"]
print(item)  # salida: cerradura
```

4. Para agregar o eliminar una clave (junto con su valor asociado), emplea la siguiente sintaxis:

```
phonebook = {}     # un diccionario vacío

phonebook["Adán"] = 3456783958     # crear/agregar un par clave-valor

print(phonebook)     # salida: {'Adán': 3456783958}

del phonebook["Adán"]

print(phonebook)     # salida: {}
```

Además, se puede insertar un elemento a un diccionario utilizando el método `update()`, y eliminar el ultimo elemento con el método `popitem()`, por ejemplo:

```
pol_esp_dictionary = {"kwiat": "flor"}

pol_esp_dictionary.update({"gleba": "tierra"})

print(pol_esp_dictionary)     # salida: {'kwiat': 'flor', 'gleba': 'tierra'}

pol_esp_dictionary.popitem()

print(pol_esp_dictionary)     # salida: {'kwiat': 'flor'}
```

5. Se puede emplear el bucle `for` para iterar a través del diccionario, por ejemplo:

```
pol_esp_dictionary = {
    "zamek": "castillo",
```

```
    "woda": "agua",
    "gleba": "tierra"
    }

for item in pol_esp_dictionary:
    print(item)

# salida: zamek
#         woda
#         gleba
```

6. Si deseas examinar los elementos (claves y valores) del diccionario, puedes emplear el método `items()`, por ejemplo:

```
pol_esp_dictionary = {
    "zamek" : "castillo",
    "woda"  : "agua",
    "gleba" : "tierra"
    }

for key, value in pol_esp_dictionary.items():
    print("Pol/Esp ->", key, ":", value)
```

7. Para comprobar si una clave existe en un diccionario, se puede emplear la palabra clave reservada `in`:

```
pol_esp_dictionary = {
    "zamek" : "castillo",
    "woda"  : "agua",
    "gleba" : "tierra"
    }

if "zamek" in pol_esp_dictionary:
    print("Si")
else:
    print("No")
```

8. Se puede emplear la palabra reservada `del` para eliminar un elemento, o un diccionario entero. Para eliminar todos los elementos de un diccionario se debe emplear el método `clear()`:

```
pol_esp_dictionary = {
    "zamek" : "castillo",
    "woda"  : "agua",
    "gleba" : "tierra"
    }

print(len(pol_esp_dictionary))     # salida: 3
del pol_esp_dictionary["zamek"]    # eliminar un elemento
print(len(pol_esp_dictionary))     # salida: 2

pol_esp_dictionary.clear()    # eliminar todos los elementos
print(len(pol_esp_dictionary))     # salida: 0
```

```
del pol_esp_dictionary    # elimina el diccionario
```

9. Para copiar un diccionario, emplea el método `copy()`:

```
pol_esp_dictionary = {
    "zamek" : "castillo",
    "woda"  : "agua",
    "gleba" : "tierra"
    }

copy_dictionary = pol_esp_dictionary.copy()
```

## Puntos Claves: Tuplas y diccionarios

**Ejercicio 1**

¿Qué ocurrirá cuando se intente ejecutar el siguiente código?

```
my_tup = (1, 2, 3)
print(my_tup[2])
```

➔ 3

**Ejercicio 2**

¿Cuál es la salida del siguiente fragmento de código?

```
tup = 1, 2, 3
a, b, c = tup
```

```
print(a * b * c)
```

➔ 6

**Ejercicio 3**

Completa el código para emplear correctamente el método `count()` para encontrar la cantidad de `2` duplicados en la tupla siguiente.

```
tup = 1, 2, 3, 2, 4, 5, 6, 2, 7, 2, 8, 9

duplicates = tup.count(2)


print(duplicates)     # salida: 4
```

**Ejercicio 4**

Escribe un programa que "una" los dos diccionarios (`d1` y `d2`) para crear uno nuevo (`d3`).

```
d1 = {'Adam Smith': 'A', 'Judy Paxton': 'B+'}

d2 = {'Mary Louis': 'A', 'Patrick White': 'C'}

d3 = {}


for item in (d1, d2):

    d3.update(item)


print(d3)
```

**Ejercicio 5**

Escribe un programa que convierta la lista `my_list` en una tupla.

```
my_list = ["car", "Ford", "flower", "Tulip"]


t = tuple(my_list)
print(t)
```

**Ejercicio 6**

Escribe un programa que convierta la tupla `colors` en un diccionario.

```
colors = (("green", "#008000"), ("blue", "#0000FF"))


colors_dictionary = dict(colors)
print(colors_dictionary)
```

**Ejercicio 7**

¿Que ocurrirá cuando se ejecute el siguiente código?

```
my_dictionary = {"A": 1, "B": 2}
copy_my_dictionary = my_dictionary.copy()
```

```
my_dictionary.clear()


print(copy_my_dictionary)
```

➔ {"A": 1, "B": 2}

**Ejercicio 8**

¿Cuál es la salida del siguiente programa?

```
colors = {
    "blanco": (255, 255, 255),
    "gris": (128, 128, 128),
    "rojo": (255, 0, 0),
    "verde": (0, 128, 0)
    }


for col, rgb in colors.items():
    print(col, ":", rgb)
```

➔ blanco : (255, 255, 255)
➔ gris : (128, 128, 128)
➔ rojo : (255, 0, 0)
➔ verde : (0, 128, 0)