Data Structures: Lists, Tuples, Sets, Dictionaries, Compound Data Structures

Lists

En Python, una lista es una estructura de datos que puede contener una colección ordenada de elementos, que pueden ser de diferentes tipos, puede ser mutable. Puedes definir una lista utilizando corchetes `[]` y separar los elementos con comas.

```
my_list = [1, 2, 3, 'cuatro', 5.0, [True, 3, "hola"]]
```

Ejemplos

```
months = ['January', 'February', 'March', 'April', 'May', 'Ju
```

```
q3 = months[6:9]
print(q3) # [ 'July', 'August', 'September']
```

```
first_half = months[:6]
print(first_half) # ['January', 'February', 'March', 'April',
```

```
second_half = months[6:]
print(second_half) # ['July', 'August', 'September', 'October
```

```
print(len(months)) # 12
```

Slice and Dice with Lists

Podemos extraer más de un valor de una lista a la vez usando el "slicing" (rebanado). Al utilizar el "slicing", es importante recordar que el índice `inferior`

es `inclusivo` y el índice `superior` es `exclusivo`.

```
>>> list_of_random_things = [1, 3.4, 'a string', True]
>>> list_of_random_things[1:2]
[3.4]
```

Operadores de Pertenencia:

Los operadores de pertenencia se utilizan para verificar si un elemento está presente en una secuencia (como una lista). Los dos operadores principales son:

`in`: Devuelve `True` si un elemento está presente en la lista.

```
result = 3 in my_list  # Devuelve True
```

`not in`: Devuelve `True` si un elemento no está presente en la lista.

```
result = 'six' not in my_list  # Devuelve True
```

Mutabilidad y Orden

La mutabilidad se refiere a si podemos cambiar un objeto una vez que ha sido creado. Si un objeto (como una lista o una cadena) puede ser modificado (como una lista puede), entonces se llama mutable. Sin embargo, si un objeto no puede ser modificado sin crear un objeto completamente nuevo (como en el caso de las cadenas), entonces se considera inmutable.

```
>>> my_list = [1, 2, 3, 4, 5]
>>> my_list[0] = 'one'
>>> print(my_list)
['one', 2, 3, 4, 5]
```

Como se mostró anteriormente, puedes reemplazar 1 con 'one' en la lista mencionada anteriormente. Esto se debe a que las `listas` son `mutables`.

Sin embargo, lo siguiente no funciona:

```
>>> greeting = "Hello there"
>>> greeting[0] = 'M'
```

Esto se debe a que las `cadenas` son `inmutables`. Esto significa que para cambiar esta cadena, necesitarás crear una cadena completamente nueva.

Hay dos cosas a tener en cuenta para cada uno de los tipos de datos que estás utilizando:

1. ¿Son mutables?
2. ¿Son ordenados?

El `orden` se refiere a si la posición de un elemento en el objeto se puede utilizar para acceder al elemento. Tanto las `cadenas` como las `listas` son `ordenadas` ordenadas. Podemos usar el orden para acceder a partes de una lista y de una cadena.

List Methods

method	definición
`append()`	Añade un elemento al final de la lista.
`clear()`	Elimina todos los elementos de la lista.
`copy()`	Devuelve una copia de la lista.
`count()`	Devuelve el número de elementos con el valor especificado.
`extend()`	Añade los elementos de una lista (o cualquier iterable) al final de la lista actual.
`index()`	Devuelve el índice del primer elemento con el valor especificado.
`insert()`	Añade un elemento en la posición especificada.
`pop()`	Quita el elemento en la posición especificada (por defecto toma la ultima).
`remove()`	Quita el primer elemento con el valor especificado.
`reverse()`	Invierte el orden de la lista.
`sort()`	Ordena la lista.

Una función útil para list

El método `join` es un método de cadena que toma una lista de cadenas como argumento y devuelve una cadena que consiste en los elementos de la lista

unidos por una cadena separadora.

```
new_str = "\n".join(["fore", "aft", "starboard", "port"])
print(new_str)
```

Output:

```
fore
aft
starboard
port
```

En este ejemplo, usamos la cadena "\n" como separador para que haya un salto de línea entre cada elemento. También podemos usar otras cadenas como separadores con `.join`. Aquí usamos un guion.

```
name = "-".join(["García", "O'Kelly"])
print(name)
```

Output:

```
García-O'Kelly
```

Tuples

Una tupla es un tipo de dato para secuencias ordenadas e inmutables de elementos. A menudo se utilizan para almacenar piezas de información relacionadas. Considera este ejemplo que involucra latitud y longitud:

```
location = (13.4125, 103.866667)
print("Latitude:", location[0])
print("Longitude:", location[1])
```

Las tuplas son similares a las listas en que almacenan una colección ordenada de objetos que se pueden acceder mediante sus índices. Sin embargo, a diferencia de las listas, las tuplas son inmutables, no puedes agregar ni quitar elementos de las tuplas ni ordenarlas.

Las tuplas también se pueden utilizar para asignar múltiples variables de manera compacta.

```
dimensions = 52, 40, 100
length, width, height = dimensions
print("The dimensions are {} x {} x {}".format(length, width,
```

Los paréntesis son opcionales al definir tuplas, y los programadores a menudo los omiten si los paréntesis no aclaran el código.

En la segunda línea, se asignan tres variables con el contenido de la tupla `dimensions`. Esto se llama desempaquetado de tuplas. Puedes utilizar el desempaquetado de tuplas para asignar la información de una tupla en múltiples variables sin tener que acceder a ellas una por una y realizar múltiples declaraciones de asignación.

Si no necesitamos usar `dimensions` directamente, podríamos acortar esas dos líneas de código en una única línea que asigna tres variables de una vez.

```
length, width, height = 52, 40, 100
print("The dimensions are {} x {} x {}".format(length, width,
```

Sets

Un conjunto (set) es un tipo de dato para colecciones mutables no ordenadas de elementos únicos. Una aplicación de un conjunto es eliminar rápidamente duplicados de una lista.

```
numbers = [1, 2, 6, 3, 1, 1, 6]
unique_nums = set(numbers)
print(unique_nums)
```

Output:

```
{1, 2, 3, 6}
```

Los conjuntos admiten el operador `in` de la misma manera que lo hacen las listas. Puedes agregar elementos a los conjuntos usando el método `add` y eliminar elementos usando el método `pop`, similar a las listas. Sin embargo, al

quitar un elemento de un conjunto, se elimina un elemento aleatorio. Recuerda que los conjuntos, a diferencia de las listas, no están ordenados, por lo que no hay un "último elemento".

```
fruit = {"apple", "banana", "orange", "grapefruit"}  # define

print("watermelon" in fruit)  # check for element

fruit.add("watermelon")  # add an element
print(fruit)

print(fruit.pop())  # remove a random element
print(fruit)
```

Output:

```
False
{'grapefruit', 'orange', 'watermelon', 'banana', 'apple'}
grapefruit
{'orange', 'watermelon', 'banana', 'apple'}
```

Otras operaciones que puedes realizar con conjuntos incluyen aquellas de conjuntos matemáticos. Métodos como unión, intersección y diferencia son fáciles de realizar con conjuntos y son mucho más rápidos que esos operadores con otros contenedores.

Dictionaries and Identity Operators

Un diccionario es un tipo de dato mutable que almacena asignaciones de claves únicas a valores. Aca tenemos un diccionario que almacena elementos y sus números atómicos.

```
elements = {"hydrogen": 1, "helium": 2, "carbon": 6}
```

Los diccionarios pueden tener claves de cualquier tipo inmutable, como enteros o tuplas, ¡no solo cadenas! Ni siquiera es necesario que todas las claves tengan el mismo tipo. Podemos buscar valores o insertar nuevos valores en el diccionario utilizando corchetes que encierran la clave.

```
print(elements["helium"])  # print the value mapped to "heliu
elements["lithium"] = 3  # insert "lithium" with a value of 3
```

Podemos verificar si un valor está en un diccionario de la misma manera que verificamos si un valor está en una lista o un conjunto con la palabra clave `in`. Los diccionarios tienen un método relacionado que también es útil, `get`. `get` busca valores en un diccionario, pero a diferencia de los corchetes, `get` devuelve `None` (o un valor predeterminado de tu elección) si la clave no se encuentra.

```
print("carbon" in elements)
print(elements.get("dilithium"))
```

Output:

```
True
None
```

El carbono está en el diccionario, por lo que imprime `True`. Dilitio no está en nuestro diccionario, por lo que `get` devuelve `None` y luego lo imprime. Si esperas que las búsquedas a veces fallen, `get` podría ser una mejor herramienta que las búsquedas normales con corchetes porque los errores pueden hacer que tu programa falle.

Operadores de Identidad

Palabra clave	Operador
`is`	evalúa si ambos lados tienen la misma identidad.
`is not`	evalúa si ambos lados tienen identidades diferentes.

Puedes verificar si una clave devuelta es `None` con el operador `is`. Puedes verificar lo opuesto usando `is not`

```
n = elements.get("dilithium")
print(n is None)
print(n is not None)
```

Output:

```
True
False
```

Compound Data Structures

Podemos incluir contenedores dentro de otros contenedores para crear estructuras de datos compuestas. Por ejemplo, este diccionario mapea claves a valores que también son diccionarios.

```
elements = {"hydrogen": {"number": 1,
                         "weight": 1.00794,
                         "symbol": "H"},
            "helium": {"number": 2,
                       "weight": 4.002602,
                       "symbol": "He"}}
```

Podemos acceder a los elementos en este diccionario anidado de la siguiente manera.

```
helium = elements["helium"]  # get the helium dictionary
hydrogen_weight = elements["hydrogen"]["weight"]  # get hydro
```

También puedes agregar una nueva clave al diccionario del elemento.

```
oxygen = {"number":8,"weight":15.999,"symbol":"O"}  # create 
elements["oxygen"] = oxygen  # assign 'oxygen' as a key to th
print('elements = ', elements)
```

Output:

```
elements =  {"hydrogen": {"number": 1,
                          "weight": 1.00794,
                          "symbol": 'H'},
             "helium": {"number": 2,
                        "weight": 4.002602,
                        "symbol": "He"},
             "oxygen": {"number": 8,
```

```
                                "weight": 15.999,
                                "symbol": "O"}}
```

Resumen

Data Structure	Ordered	Mutable	Constructor	Example
Lista	Si	Si	`[ ]` or `list()`	`[5.7, 4, 'yes', 5.7]`
Tupla	Si	No	`(` `)` or `tuple()`	`(5.7, 4, 'yes', 5.7)`
Set	No	Si	`{}`* or `set()`	`{5.7, 4, 'yes'}`
Diccionario	No	No**	`{ }` or `dict()`	`{'Jun': 75, 'Jul': 89}`

Se puede utilizar llaves para definir un conjunto de esta manera: {1, 2, 3}. Sin embargo, si dejas las llaves vacías como esto: `{}` Python creará en su lugar un diccionario vacío. Así que, para crear un conjunto vacío, utiliza `set()`.

Un diccionario en sí mismo es mutable, pero cada una de sus claves individuales debe ser inmutable.