Hoja de trucos: Estructuras de datos de Python Parte-2

Diccionarios

Paquete/Método	Descripción	Ejemplo de código

Creating a Dictionary	Un diccionario es un tipo de dato incorporado que representa una colección de pares clave-valor. Los diccionarios están encerrados en llaves `{}`.	Ejemplo:
		`dict_name = {} #Crea un diccionario vacío`
		`person = { "name": "John",  "age": 30, "city": "New York"}`

Accediendo a valores	Puedes acceder a los valores en un diccionario usando sus correspondientes `keys`.	Sintaxis:
		`Value = dict_name["key_name"]`
		Ejemplo:
		`name = person["name"]`
		`age = person["age"]`

Agregar o modificar	Inserta un nuevo par clave-valor en el diccionario. Si la clave ya existe, el valor se actualizará; de lo contrario, se crea una nueva entrada.	Sintaxis:
		`dict_name[key] = value`
		Ejemplo:
		`person["Country"] = "USA" # Se creará una nueva entrada.`
		`person["city"] = "Chicago"  # Actualiza el valor existente para la misma clave`

del	Elimina el par clave-valor especificado del diccionario. Lanza un `KeyError` si la clave no existe.	Sintaxis:
		`del dict_name[key]`
		Ejemplo:
		`del person["Country"]`

update()	El método `update()` fusiona el diccionario proporcionado en el diccionario existente, agregando o actualizando pares clave-valor.	Sintaxis:
		`dict_name.update({key: value})`
		Ejemplo:
		`person.update({"Profession": "Doctor"})`

clear()	El método `clear()` vacía el diccionario, eliminando todos los pares clave-valor dentro de él. Después de esta operación, el diccionario sigue siendo accesible y se puede usar más adelante.	Sintaxis:
		`dict_name.clear()`
		Ejemplo:
		`grades.clear()`

existencia de clave	Puedes comprobar la existencia de una clave en un diccionario usando la palabra clave `in`.	Ejemplo:
		`if "name" in person:`
		`  print("El nombre existe en el diccionario.")`

copy()	Crea una copia superficial del diccionario. El nuevo diccionario contiene los mismos pares clave-valor que el original, pero permanecen como objetos distintos en memoria.	Sintaxis:
		`new_dict = dict_name.copy()`
		Ejemplo:
		`new_person = person.copy()`
		`new_person = dict(person) # otra forma de crear una copia del diccionario`

keys()	Recupera todas las claves del diccionario y las convierte en una lista. Útil para iterar o procesar claves usando métodos de lista.	Sintaxis:
		`keys_list = list(dict_name.keys())`
		Ejemplo:
		`person_keys = list(person.keys())`

values()	Extrae todos los valores del diccionario y los convierte en una lista. Esta lista se puede usar para	Sintaxis:

	procesamiento o análisis posterior.	`values_list = list(dict_name.values())`
		Ejemplo:
		`person_values = list(person.values())`

items()	Recupera todos los pares clave-valor como tuplas y las convierte en una lista de tuplas. Cada tupla consiste en una clave y su correspondiente valor.	Sintaxis:
		`items_list = list(dict_name.items())`
		Ejemplo:
		`info = list(person.items())`

Conjuntos

Paquete/Método	Descripción	Ejemplo de código

add()	Los elementos se pueden agregar a un conjunto usando el método `add()`. Los duplicados se eliminan automáticamente, ya que los conjuntos solo almacenan valores únicos.	Sintaxis:
		`set_name.add(element)`
		Ejemplo:
		`fruits.add("mango")`

clear()	El método `clear()` elimina todos los elementos del conjunto, resultando en un conjunto vacío. Actualiza el conjunto en su lugar.	Sintaxis:
		`set_name.clear()`
		Ejemplo:
		`fruits.clear()</td>`

copy()	El método `copy()` crea una copia superficial del conjunto. Cualquier modificación a la copia no afectará al conjunto original.	Sintaxis:
		`new_set = set_name.copy()`
		Ejemplo:
		`new_fruits = fruits.copy()`

Definiendo Conjuntos	Un conjunto es una colección desordenada de elementos únicos. Los conjuntos están encerrados en llaves `{}`. Son útiles para almacenar valores distintos y realizar operaciones de conjuntos.	Ejemplo:
		`empty_set = set() #Creando un conjunto vacío`
		`Set fruits = {"apple", "banana", "orange"}`

discard()	Usa el método `discard()` para eliminar un elemento específico del conjunto. Ignora si el elemento no se encuentra.	Sintaxis:
		`set_name.discard(element)`
		Ejemplo:
		`fruits.discard("apple")`

issubset()	El método `issubset()` verifica si el conjunto actual es un subconjunto de otro conjunto. Devuelve True si todos los elementos del conjunto actual están presentes en el otro conjunto, de lo contrario False.	Sintaxis:
		`is_subset = set1.issubset(set2)`
		Ejemplo:
		`is_subset = fruits.issubset(colors)`

issuperset()	El método `issuperset()` verifica si el conjunto actual es un superconjunto de otro conjunto. Devuelve True si todos los elementos del otro conjunto están presentes en el conjunto actual, de lo contrario False.	Sintaxis: is_superset = set1.issuperset(set2) Ejemplo: `is_superset = colors.issuperset(fruits)`

pop()	El método `pop()` elimina y devuelve un elemento arbitrario del conjunto. Lanza un `KeyError` si el conjunto está vacío. Usa este método para eliminar elementos cuando el orden no importa.	Sintaxis:
		`removed_element = set_name.pop()`
		Ejemplo:
		`removed_fruit = fruits.pop()`

remove()	Usa el método `remove()` para eliminar un elemento específico del conjunto. Lanza un `KeyError` si el elemento no se encuentra.	Sintaxis:
		`set_name.remove(element)`
		Ejemplo:
		`fruits.remove("banana")`

Operaciones de Conjuntos	Realiza varias operaciones en conjuntos: `unión`, `intersección`, `diferencia`, `diferencia simétrica`.	Sintaxis:
		`union_set = set1.union(set2)`
		`intersection_set = set1.intersection(set2)`
		`difference_set = set1.difference(set2)`
		`sym_diff_set = set1.symmetric_difference(set2)`
		Ejemplo:
		`combined = fruits.union(colors)`
		`common = fruits.intersection(colors)`
		`unique_to_fruits = fruits.difference(colors)`
		`sym_diff = fruits.symmetric_difference(colors)`

update()	El método `update()` agrega elementos de otro iterable al conjunto. Mantiene la unicidad de los elementos.	Sintaxis:
		`set_name.update(iterable)`
		Ejemplo:
		`fruits.update(["kiwi", "grape"])`

Skills Network