|  |
| --- |
|  |
| 1) A tuple is a sequence of values much like a list. The values stored in a tuple can be any type, and they are indexed by integers. The important difference is that tuples are immutable . |
|  | t2[0]= 34 |
|  | output: |
|  | TypeError Traceback (most recent call last) |
|  | Cell In[23], line 1 |
|  | ----> 1 t2[0]= 34 |
|  |  |
|  | TypeError: 'tuple' object does not support item assignment |
|  | this shows that tuples are immutable |
|  |  |
|  | 2) The two methods are count() and index() |
|  | The count() method of Tuple returns the number of times the given element appears in the tuple. |
|  | The Index() method returns the first occurrence of the given element from the tuple. |
|  |  |
|  | Syntax: |
|  |  |
|  | tuple.index(element, start, end) |
|  |  |
|  |  |
|  | 3) he set datatype in python does not allow duplicate items. |
|  |  |
|  | Code to remove duplicates from the given list using a set: |
|  |  |
|  | python |
|  | lst = [1, 1, 1, 2, 1, 3, 1, 4, 2, 1, 2, 2, 2, 3, 2, 4, 3, 1, 3, 2, 3, 3, 3, 4, 4, 1, 4, 2, 4, 3, 4, 4] |
|  | lst = list(set(lst)) |
|  | print(lst) |
|  | Output: |
|  | [1, 2, 3, 4] |
|  |  |
|  | 4)The union() method of a set returns a new set that contains all the elements from the original set and the set(s) passed as an argument, without any duplicates. This method does not modify the original set. |
|  |  |
|  | Example: |
|  |  |
|  | python |
|  | set1 = {1, 2, 3} |
|  | set2 = {3, 4, 5} |
|  | set3 = set1.union(set2) |
|  | print(set3) # Output: {1, 2, 3, 4, 5} |
|  | The update() method of a set adds all the elements from the set(s) passed as an argument to the original set. This method modifies the original set. |
|  |  |
|  | Example: |
|  |  |
|  | python |
|  | set1 = {1, 2, 3} |
|  | set2 = {3, 4, 5} |
|  | set1.update(set2) |
|  | print(set1) # Output: {1, 2, 3, 4, 5} |
|  |  |
|  | 5) In Python, a dictionary is an unordered collection of key-value pairs, where each key is unique. It is also known as an associative array, hash table, or a map. |
|  |  |
|  | Example: |
|  |  |
|  | python |
|  | my\_dict = {'name': 'John', 'age': 25, 'city': 'New York'} |
|  | print(my\_dict) |
|  | Output: |
|  | {'name': 'John', 'age': 25, 'city': 'New York'} |
|  | In this example, we have created a dictionary named `my\_dict` with three key-value pairs: `name: John`, `age: 25`, and `city: New York`. |
|  | As mentioned earlier, dictionaries are unordered, which means that the order of the elements is not guaranteed. However, starting from Python 3.7, the insertion order of elements is preserved in dictionaries. |
|  |  |
|  | 6) Yes, we can create a nested dictionary in Python. A nested dictionary is a dictionary inside another dictionary, where each value is also a dictionary. |
|  |  |
|  | Example: |
|  |  |
|  | python |
|  | my\_dict = { |
|  | 'person1': { |
|  | 'name': 'John', |
|  | 'age': 25, |
|  | 'city': 'New York' |
|  | }, |
|  | 'person2': { |
|  | 'name': 'Jane', |
|  | 'age': 30, |
|  | 'city': 'London' |
|  | } |
|  | } |
|  |  |
|  | print(my\_dict) |
|  |  |
|  |  |
|  | Output: |
|  |  |
|  | {'person1': {'name': 'John', 'age': 25, 'city': 'New York'}, |
|  | 'person2': {'name': 'Jane', 'age': 30, 'city': 'London'}} |
|  |  |
|  |  |
|  | In this example, we have created a dictionary `my\_dict` with two keys `person1` and `person2`. The value of each key is another dictionary with three keys `name`, `age`, and `city`. This is a simple one-level nested dictionary. |
|  |  |
|  | 7) Here's an example of how to use the `setdefault()` method to create a key named `topics` in a dictionary and set its value to `['Python', 'Machine Learning’, 'Deep Learning']`: |
|  |  |
|  | python |
|  | my\_dict = {'name': 'John', 'age': 25} |
|  |  |
|  | # Using setdefault() to create a key 'topics' with value ['Python', 'Machine Learning', 'Deep Learning'] |
|  | my\_dict.setdefault('topics', ['Python', 'Machine Learning', 'Deep Learning']) |
|  | print(my\_dict) |
|  |  |
|  | Output: |
|  | {'name': 'John', 'age': 25, 'topics': ['Python', 'Machine Learning', 'Deep Learning']} |
|  |  |
|  | In this example, we have used the `setdefault()` method to create a key named `topics` in the `my\_dict` dictionary and set its value to `['Python', 'Machine Learning’, 'Deep Learning']`. Since the `topics` key did not exist in the dictionary before, it was created and its value was set to the default value provided as the second argument to the `setdefault()` method. |
|  |  |
|  | 8) In Python, dictionaries have three view objects: |
|  |  |
|  | 1. `dict.keys()` - returns a view object that contains the keys of the dictionary. |
|  | 2. `dict.values()` - returns a view object that contains the values of the dictionary. |
|  | 3. `dict.items()` - returns a view object that contains the key-value pairs of the dictionary as tuples. |
|  |  |
|  | Here's an example of how to use these three in-built methods to display the view objects for a given dictionary: |
|  |  |
|  | python |
|  | my\_dict = {'name': 'John', 'age': 25, 'city': 'New York'} |
|  |  |
|  | # Displaying the keys view object |
|  | print(my\_dict.keys()) |
|  |  |
|  | # Displaying the values view object |
|  | print(my\_dict.values()) |
|  |  |
|  | # Displaying the items view object |
|  | print(my\_dict.items()) |
|  |  |
|  |  |
|  | Output: |
|  |  |
|  | dict\_keys(['name', 'age', 'city']) |
|  | dict\_values(['John', 25, 'New York']) |
|  | dict\_items([('name', 'John'), ('age', 25), ('city', 'New York')]) |
|  |  |
|  |  |
|  | In this example, we have displayed the three view objects for the `my\_dict` dictionary using the `keys()`, `values()`, and `items()` in-built methods. The `keys()` method returns a view object that contains the keys of the dictionary, the `values()` method returns a view object that contains the values of the dictionary, and the `items()` method returns a view object that contains the key-value pairs of the dictionary as tuples. |
|  |  |