Programming Fundamentals and Python

Data types - Tuple

Tuple

A tuple is a container which holds a series of comma-separated values (items or elements) between parentheses such as an (x, y) co-ordinate. Tuples are like lists, except they are immutable (i.e. you cannot change its content once created) and can hold mix data types. Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.

A tuple is a collection which is ordered and unchangeable.

Tuples are written with round brackets.

Creating a Tuple

A tuple can be written as the collection of comma-separated (,) values enclosed with the small () brackets. The parentheses are optional but it is good practice to use. A tuple can be defined as follows. An empty tuple can be created.

```
>>> T1 = (101, "Peter", 22)
>>> T2 = ("Apple", "Banana", "Orange")
>>> T3 = 10,20,30,40,50
>>> print(type(T1))
<class 'tuple'>
>>> print(type(T2))
<class 'tuple'>
>>> print(type(T3))
<class 'tuple'>
>>> T4=()
```

Items and Index

Tuple items are ordered, unchangeable, and allow duplicate values.
Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

tuple() constructor

It is also possible to use the tuple() constructor to make a tuple.
NOTE: the double round brackets.

```
>>> T=tuple(("apple", "banana", "cherry"))
>>> print(type(T1))
<class 'tuple'>
>>> T=tuple("apple", "banana", "cherry")
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    T=tuple("apple", "banana", "cherry")
TypeError: tuple expected at most 1 argument, got 3
>>>
```

Update Tuple

Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created. Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

```
>>> L=list(T1)
>>> L
['apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'mango']
>>> print (type(L))
<class 'list'>
>>> x=tuple(L)
>>> x
('apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'mango')
>>> print(type(x))
<class 'tuple'>
>>>
```

Add Items

Since tuples are immutable, they do not have a build-in append() method, but there are other ways to add items to a tuple.

1. Convert into a list:

Just like the workaround for changing a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

CODE

```
T1=("apple", "banana", "cherry", "kiwi", "melon", "mango")
y = list(T1)
y.append("orange")
T1 = tuple(y)
```

OUTPUT

```
>>> T1
('apple', 'banana', 'cherry', 'kiwi', 'melon', 'mango')
>>>
==== RESTART: C:\Users\Hera Noor\AppData\Local\Programs\Python\Pyth
>>> T1
('apple', 'banana', 'cherry', 'kiwi', 'melon', 'mango', 'orange')
>>> T1
```

Data Type – Sets

Sets

- Sets are used to store multiple items in a single variable.
- A set is a collection which is unordered, Unchangeable, unindexed and does not allows duplicate values.

- Unordered means that the items in a set do not have a defined order.
- Unchangeable, meaning that we can change the items after the set has been created but we can add new items or remove the existing ones.
- Sets cannot have two items with the same value.

- Most of the method like len(), type() are same as discussed in the other data types.
- The Set() constructor are used to convert it into the set data type.

Method	Description
add()	Adds an element to the set
update()	Update the set with the union of this set and others
difference()	Returns a set containing the difference between two or more sets
remove()	Removes the specified element, Raise error if element does not exist in set
discard()	Remove the specified item, No error occur if item doesn't exist
pop()	Removes the first element from the set. *(No indexing allowed in it)*
union()	Return a set containing the union of sets
intersection()	Returns a set, that is the intersection of two other sets

Method	Example

add()	`thisset = {"apple", "banana", "cherry"}`
	`thisset.add("orange")`
	`print(thisset)`
	
	`{'orange', 'apple', 'banana', 'cherry'}`

update()	`x = {"apple", "banana", "cherry"}`
	`y = {"google", "microsoft", "apple"}`
	`x.update(y)`
	`print(x)`
	
	`{'banana', 'apple', 'cherry', 'google', 'microsoft'}`

Method	Example

difference()	`x = {"apple", "banana", "cherry"}`
	`y = {"google", "microsoft", "apple"}`
	`z = x.difference(y)`
	`print(z)`
	
	`{'banana', 'cherry'}`

remove()	`fruits = {"apple", "banana", "cherry"}`
	`fruits.remove("banana")`
	`print(fruits)`
	
	`{'apple', 'cherry'}`
	
	`fruits = {"apple", "banana", "cherry"}`
	`fruits.remove("orange")`
	`print(fruits)`
	
	`Traceback (most recent call last):`
	`  File "./prog.py", line 2, in <module>`
	`KeyError: 'orange'`

Method	Description

discard()	`fruits = {"apple", "banana", "cherry"}`
	`fruits.discard("banana")`
	`print(fruits)`
	`{'cherry', 'apple'}`
	
	`fruits = {"apple", "banana", "cherry"}`
	`fruits.discard("orange")`
	`print(fruits)`
	`{'banana', 'cherry', 'apple'}`

pop()	`fruits = {"apple", "banana", "cherry"}`
	`fruits.pop()`
	`print(fruits)`
	`{'banana', 'cherry'}`

Method	Description

union()	`x = {"apple", "banana", "cherry"}`
	`y = {"google", "cherry", "apple"}`
	`z = x.union(y)`
	`print(z)`
	`{'banana', 'cherry', 'google', 'apple'}`

intersection()	`x = {"apple", "banana", "cherry"}`
	`y = {"google", "cherry", "apple"}`
	`z = x.intersection(y)`
	`print(z)`
	`{'cherry', 'apple'}`

Data types - Dictionaries

Python Dictionaries

Dictionary in Python is a collection of keys values, used to store data values like a map, which, unlike other data types which hold only a single value as an element. The dictionary is defined into element Keys and values.

- Keys must be a single element
- Value can be any type such as list, tuple, integer, etc.

A dictionary is an associative array (also known as hashes). Any key of the dictionary is associated (or mapped) to a value. The values of a dictionary can be any Python data type. So dictionaries are unordered key-value-pairs.

Properties

- **Accessed by key, not offset position**

Dictionaries are sometimes called associative arrays or hashes. They associate a set of values with keys, so an item can be fetched out of a dictionary using the key under which it is originally stored. The same indexing operation can be utilized to get components in a dictionary as in a list, but the index takes the form of a key, not a relative offset.

- **Unordered collections of arbitrary objects**

Unlike in a list, items stored in a dictionary aren‘t kept in any particular order. Keys provide the symbolic (not physical) locations of items in a dictionary.

- **Variable-length, heterogeneous, and arbitrarily nestable**

Like lists, dictionaries can grow and shrink in place (without new copies being made), they can contain objects of any type, and they support nesting to any depth (they can contain lists, other dictionaries, and so on). Each key can have just one associated value, but that value can be a collection of multiple objects if needed, and a given value can be stored under any number of keys.

- **Of the category "mutable mapping"**

Dictionary allows in place changes by assigning to indexes (they are mutable), but they don't support the sequence operations that work on strings and lists. Because dictionaries are unordered collections, operations that depend on a fixed positional order (e.g., concatenation, slicing) don't make sense.

Instead, dictionaries are the only built-in, core type representatives of the mapping category— objects that map keys to values. Other mappings in Python are created by imported modules.

- **Tables of object references (hash tables)**

If lists are arrays of object references that support access by position, dictionaries are unordered tables of object references that support access by key. Internally, dictionaries are implemented as hash tables (data structures that support very fast retrieval), which start small and grow on demand. Moreover, Python employs optimized hashing algorithms to find keys, so retrieval is quick. Like lists, dictionaries store object references (not copies, unless explicitly asked).

Create a Dictionary

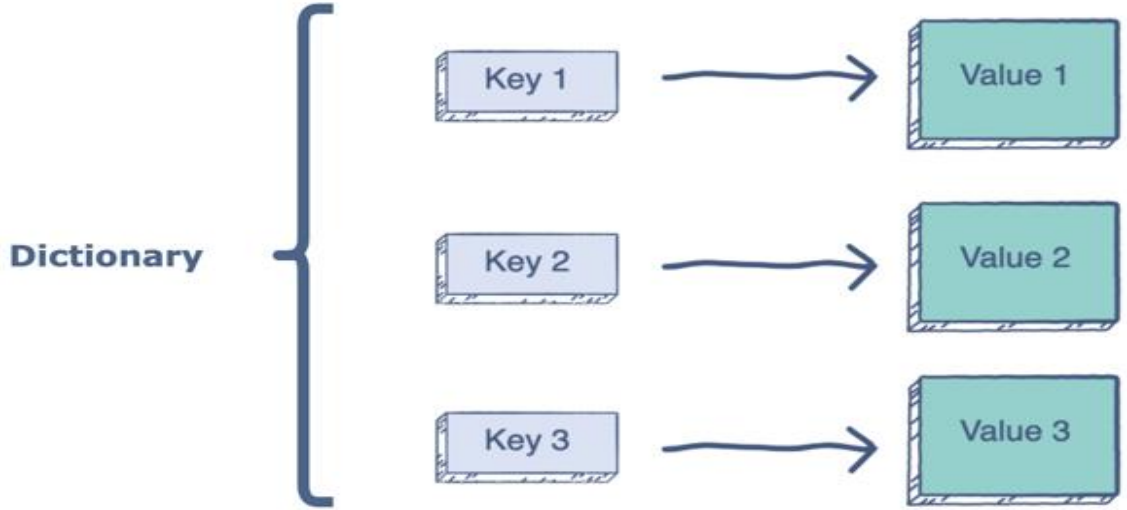
In Python, a dictionary can be created by placing a sequence of elements within curly {} braces, separated by 'comma'. Dictionary holds pairs of values, one being the Key and the other corresponding pair element being its Key value. Dictionary can also be created by the built-in function dict( ).

Note – Dictionary keys are case sensitive, the same name but different cases of Key will be treated distinctly.

```
>>> dict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>> dict
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
>>>
```

How does the dictionary works in Python?

Each key is associated with a single value. The association of a key and a value is called a key-value pair or an item. Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type (i.e., strings, numbers, or tuples).



Dictionary Item

The values in dictionary items can be of any data type. Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

```
>>> dict = {
  "1": "apple",
  "2": False,
  "3": 1964
}
>>> dict
{'1': 'apple', '2': False, '3': 1964}
>>> print(dict['1'])
apple
>>>
```

When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change. Unordered means that the items does not have a defined order, you cannot refer to an item by using an index. **Dictionaries cannot have two items with the same key.**

Dictionary Length

To determine how many items a dictionary has, use the **len()** function:

```
>>> dict
{'1': 'apple', '2': 'mani', '3': 1964}
>>> print(dict['1'])
apple
>>> print(len(dict))
3
>>>
```

type()

From Python's perspective, dictionaries are defined as objects with the data type 'dict‘.

```
>>> print(type(dict))
<class 'dict'>
>>>
```

Access

1. The items of a dictionary can be accessed by referring to its key name, inside square brackets.
2. Another method called to access is **get()** that will give you the same result.

```
>>> dict
{'1': 'corolla', '2': 'alto', '3': 'farrari', '4': 'altis', '5': 'cultus', '6':
'city', '7': '1993'}
>>> x=dict['5']
>>> x
'cultus'
>>> y=dict.get('4')
>>> y
'altis'
>>>
```