TOOL

Built-in Python Container Data Types

Containers are data types that hold other pieces of data but differ in how those data elements are accessed and what sorts of operations they support.

Lists

- A Python list is a mutable sequence of objects; list elements can be of any type.
- List elements are indexed by their position in the sequence, with valid indices being integers starting with 0 and ending with n-1, where n is the number of elements in the list.
- List elements are accessed using square brackets (`[ ]`) and can be used for either getting or setting elements at a given position; e.g.,
  - `my_list[2]` returns the element at position 2 in my_list.
  - `my_list[3] = 4` sets the element at position 3 in my_list to the value 4.
- Lists can also be sliced using square brackets, to get contiguous blocks of elements.
  - `my_list[i:j]` returns a list containing elements from my_list starting at position i and ending at position j-1 (so that the number of elements in the sublist is j-i).
- Lists are mutable; elements can be changed, added, or removed.
- for loops are a useful way of iterating over all the elements of a list; e.g.,

```
for elem in my_list:
    # elem will be set to each successive element in my_list
```

- Lists can be created in various ways, including list comprehensions:
  `[expression for element in iterable if condition]`
- Some useful list methods include:
  - `list.append(obj):` add an object to the end of a list.
  - `list.extend(alist):` add the elements of a list to the end of a list.
  - `list.insert(index, obj):` insert an object into a list at the specified index.
  - `list.count(obj):` count the number of times a given object appears in a list.
  - `list.index(obj):` return the index of the first occurrence of an object in a list.
  - `list.sort():` sort the elements of a list (modifying the original list).

Dictionaries

- A Python dictionary maps a set of keys to an associated set of values.
- Dictionary elements are accessed by their key; if a specified key does not exist in a dictionary, an error is thrown.

- Dictionary values are accessed by their key using square brackets (`[ ]`) and can be used for getting, setting, or adding new elements for a given key; e.g.,
  - `my_dict['a']` returns the value in my_dict associated with the key 'a' .
  - `my_dict['b'] = 5` sets the value in my_dict associated with the key 'b' to the value 5, or adds it to the dictionary.
- Dictionaries are mutable; key-value pairs can be changed, added, or removed.
- for loops are a useful way of iterating over all the key-value pairs (items) in a dictionary; e.g.,

```
for k,v in my_dict.items():
    # k will be set to each key in my_dict
    # v will be set to the value associated with k
```

- Dictionaries can be created in various ways, including dictionary comprehensions:
  `{k:v for element in iterable if condition}`
- Some useful dictionary methods include:
  - `dict.keys()`: return an iterable view of all keys in a dictionary.
  - `dict.values()`: return an iterable view of all values in a dictionary.
  - `dict.items()`: return an iterable view of all key-value pairs in a dictionary.
  - `dict.get(key, default)`: return the value associated with a specified key, return default value if key not in dictionary.

Sets

- A Python set is a mutable, unordered collection of unique elements.
- for loops are a useful way of iterating over all the elements of a set; e.g.,

```
for elem in my_set:
    # elem will be set to each successive element in my_set
    # elements of my_set will be accessed in arbitrary order
```

Sets can be created in various ways, including set comprehensions:

`{expression for element in iterable if condition}`

- Some useful set methods include:
  - `set.add(object)`: add object to a set.
  - `set.union(set2)`: return the union between set and set2.
  - `set.intersection(set2)`: return the intersection between set and set2.
  - `set - set2`: return all elements in set but not in set2 (equivalent to set.difference(s2)).

Tuples

A Python tuple is an immutable sequence.

- Tuples are indexed and sliced like lists, using square brackets, but only for getting elements (elements cannot be set once a tuple is created).

- Tuples are useful for bundling together a group of different data elements, distinguishing each element by its position in the tuple.
- Tuples are useful for unpacking multiple objects returned by a function; this requires that the function return as many objects as are assigned to on the left-hand side; e.g.,

```
x,y,z = my_func()
# requires that my_func() return three objects, or a tuple with three elements
```