CSE231 - Lab 08

Dictionaries

Dictionaries!

Probably the most useful data structure in Python. With dictionaries, you define key-value pairs of objects, and are able to extract the value from the defined key. Note that you are *not* able to call by index.

A key must be *immutable*, whereas a value can be either *mutable or immutable*. Each key must also be *unique*. (Although there are special dictionaries in a module we'll talk about later)

Mutable types include: `list`, `dict`, `set`

Immutable types include: `int`, `float`, `str`, `tuple`

Similar Concepts

List

index	value
0	"Eggs"
1	"Milk"
2	"Cheese"
3	"Yogurt"
4	"Butter"
5	"More Cheese"

Dictionary

key	value
'Eggs'	2.59
'Milk'	3.19
'Cheese'	4.80
'Yogurt'	1.35
'Butter'	2.59
'More Cheese'	6.19

Basic Example

```
D = dict()

D[‘key’] = ‘value’

print(D)    # { ‘key’: ‘value’ }

print( D[“key”] )    # ‘value’
```

Here, we’re creating a dictionary with a string-key and a string-value. You create new keys with the ‘[ ]’ notation, and you can assign a value to a key with the `=` operator. You can update the value associated with a certain key in a similar style.

Initializations

```
D = {}    # Using curly braces

D = dict()    # Using the type-function

D = { 200: “EE”, 100: “ME”}

# ^ Using curly braces + values/keys

D = dict( [(“a”, 1), (“b”, 2), (“c”, 3)] )

# You can also initialize a dictionary using a list

# of tuples, keep this in mind for next time!
```

Adding to a Dictionary

If you were to try and access a key that doesn't exist inside your dictionary, you would get a `KeyError.` It's important to check if the key exists by using the `` `in` `` keyword.

In the `` `collections` `` module, there is also a dictionary-type called a `` `defaultdict` `` that will create keys if you try to call one that doesn't exist. This would usually throw a `KeyError` with normal dictionaries. Not something you'll be tested on, but can come in handy for projects.

L8-1

Dictionary Methods

`.items()` - Returns a `dict_items()` list of all key-value pairs in a list-of-tuples-like container

`.keys()` - Returns a `dict_keys()` list of all keys

`.values()` - Returns a `dict_values()` list of all values

`.pop(key)` - Returns the value associated with the given `key`, and removes the key-value pair

`.get(key)` - Returns the value if the `key` exists, else returns `None`

del

If you want to remove an entire key-value pair without a return, you can use the `del` keyword. Will raise `KeyError` if the key doesn't exist, which can be helpful if you want your program to crash or catch.

```
D = { “a”: 1, “b”: 2 }

del D[“a”]

print(D)    # {“b”: 2}
```

Iterating Through a Dictionary

There are many, many ways to iterate through a dictionary. Dictionaries are ordered (the textbook says unordered due to its age, dictionaries became ordered in a recent version of Python), and so the iteration order will follow the way the dictionary is currently structured just as lists and tuples behave during iteration.

The method functions we talked about earlier are pretty commonly used in for-loops to iterate through a certain subset of the dictionary.

Sorting a Dictionary

Dictionaries *do not* have a `.sort()` method, so instead, you have to use the `sorted()` function. Remember the dictionary method functions we talked about earlier? We can use those to our advantage when trying to sort a dictionary!

It ends up being nearly identical to sorting containers we've already been working with. You can think of dictionaries as just secretly being lists of tuples, where each tuple contains your `key` at index 0, and `value` at index 1.