DS2000
Fall 2021
Handout: Dictionaries

A dictionary is a collection of key/value pairs that are grouped together under a single variable name. It's a lot like a list — in particular, both are variables that hold multiple values, and both are **mutable.** The main difference is that, instead of position, values in a dictionary are accessed by their key.

Here's a **list**, each item has a *position* and corresponding value:

Position	0	1	2	3
Value	90	85	92	89

And a similar **dictionary**, each item has a *key* and corresponding value:

Key	'hw1'	'hw2'	'hw3'	'hw4'
Value	90	85	92	89

Values can be repeated in a dictionary. The below dictionary is totally fine :):):)

Key	'hw1'	'hw2'	'hw3'	'hw4'
Value	90	90	90	90

Keys cannot be repeated in a dictionary. The below dictionary is NOT FINE (Python won't even let this exist) :(:(:(:(:(

Key	'hw1'	'hw1'	'hw1'	'hw1'
Value	90	85	82	89

Although we don't use positions in dictionaries, Python still respects the order of your items. As you put new key/value pairs into a dictionary, they will be added to the end, just like appending to a list. When you iterate through a dictionary, keys/values will appear in the same order they were added. (Fun fact: this guaranteed-ordering is a recent feature in Python. If we were using Python 3.1, it wouldn't be true!)

Consider an *actual* dictionary, containing translations from English words to Pirate words:

```
pirate = {'hello':'ahoy', 'wine':'grog', 'hey':'avast', 'take':'plunder'}
```

This statement creates 4 key/value pairs. An English word is a *key*, and its Pirate translation is the *value*.

Key	'hello'	'wine'	'hey'	'take'
Value	'ahoy'	'grog'	'avast'	'plunder'

From here on in, we can treat the dictionary *almost* like a list, except we use a key instead of a position to access a specific value:

```
print(pirate['hello'])  # prints 'ahoy'
print(pirate['wine'])   # prints 'grog'
print(pirate['delightful']) # KeyError, similar to IndexError
```

Common Dictionary Operations

What do we do with a list once it exists in our program? Certainly it depends on the data and your application, but here are a few common operations:

1. ***Populate a dictionary with user input.*** The values in the list might be typed in by you, the programmer, but they're more likely to come from somewhere else, such as the user typing at the prompt or from a file. Say we want to the user to give us English/Pirate words for the list. We start by declaring an empty dictionary, and then use a for loop to prompt the user.

```
pirate = {}
while condition:
      eng = input('Enter an English word\n')
      pr = input('OK, and the same thing in Pirate?\n')
      pirate[eng] = pr
```

2. ***Find out if a specific key or value exists in the dictionary.*** The input came from the user in this case, and maybe we want to know if they put in a specific word. We can use *in* like we do with lists, and that will search the keys of the dictionary. We can also search the values of the dictionary, but we need the dictionary's **`values()`** function.

```
if 'hello' in pirate: # search keys
      print('That key exists in the dictionary!')

if 'bonjour' in pirate.values(): # search values
      print('That value exists in the dictionary!')
```

3. ***Iterate through the dictionary***. Once a dictionary is populated, you might want to inspect every key/value pair in the list. For example, maybe we want to print every English word (key) and its Pirate equivalent (value). We use Python's for/in loop to do this, along with the dictionary's **`items()`** function.

```
for key, value in pirate.items():
        print('English:', key, 'Pirate:', value)
```

4. ***Remove a key/value pair from the dictionary.*** Sticking with the translation example, we might want to remove a word we find objectionable or useless. You might remember that we do a removal in a couple of different ways for a list, but in a dictionary we tend to care more about the key than the value, so that's what we use:

```
del pirate['wine']            # remove wine / grog
```

More Dictionary Operations

We use square brackets to access individual items within the list, always via the *key*. For example if you want to create the key/value pair friend / matey, you would do this:

```
mylist['friend'] = 'matey'
```

And recall that keys must be distinct, so the following statement would overwrite friend/ami.

```
mylist['friend'] = 'hearty'
```

Similarly, if you wanted to print the value with key 'friend', you would use this Python statement:

```
print(pirate['friend'])
```

If you try either of these and there is no value with that key, you'll get a **`KeyError`**, which is similar to a list's IndexError. It's an error because you're trying to access a key that's not part of your dictionary.

```
len(mydict)
```

Returns the length of the dictionary. Does not modify the dictionary.

```
mydict.get('friend')
```

Returns the value for key 'friend'. Does not modify the dictionary. Similar to `mydict['friend']` but does not result in a KeyError if the key friend doesn't exist in the dictionary.

```
for key in mydict:
    print(key)
```

Prints all the keys, and no values. Does not modify the dictionary.

```
for key in mydict:
    print(mydict[key])
```

Prints all the values, and no keys. Does not modify the dictionary.

```
mydict.clear()
```

Removes all keys and values from the dictionary.

```
mydict.items()
```

Returns all the key/value pairs. Does not modify the dictionary.