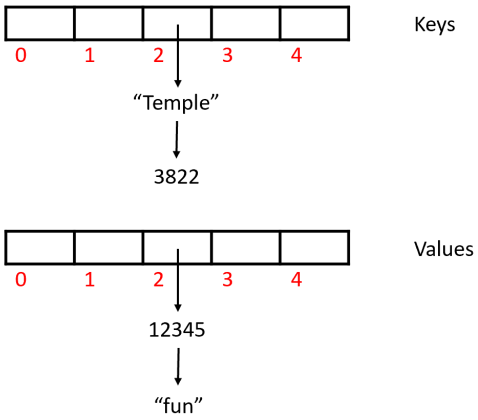
Project 5

For this project, we created our own version of a Python dictionary. Our dictionary object consists of an array of keys, as well as an array of values. These arrays function as hash tables: they're fixed in size, and the insertion location is determined with the help of a hash function. Additionally, linked lists were used to resolve all collisions.

How the dictionary works



The figure above presents an example of how the keys and values are organized in the dictionary. The index of a key in the array of keys is the same as the index of the respective value in the array of values. For example, the key-value pairs of the dictionary displayed above are ('Temple', 12345) and (3822, 'fun').

When a key-value pair is **inserted** into the dictionary, the hash function returns a location index based on the key, and the key is then appended to the linked list of that location in the array of keys. The value is then appended to the linked list that is located on the same index in the array of values.

When **retrieving** a value from a given key, the hash function calculates the location of the key in the array of keys, and the program goes to retrieve the corresponding value in the array of values by using the same location index.

The `dictionary` class has four main attributes:

- `keys` – the array of keys
- `values` – the array of values
- `size` – the size of the arrays of keys and values
- `length` – the number of key-value pairs in the dictionary

The dictionary accepts only unique keys which can be integers, floats or strings. Inserting any other key type will result in a `KeyError`.

Creating a dictionary object

To create a dictionary object, import the `dictionary` class from `data_structures.py`. The dictionary object takes the size of the arrays of keys and values as its only input argument. By default, the size of the arrays is `10`. The default size is intentionally kept small, so that the user can observe collisions easily if they would like to.

```
from data_structures import dictionary
myDictionary = dictionary(size=10)
```

Inserting a key-value pair

There are two ways to insert a key-value pair into the dictionary:

```
myDictionary.insert(key=3822, value="fun")          # first method
myDictionary[3822] = "fun"                          # second method
```

Retrieving a value from a given key

There are two ways to retrieve a value from an existing dictionary key:

```
value = myDictionary.get(key=3822)                  # first method
value = myDictionary[3822]                          # second method
```

Printing the dictionary

There are two ways to print the dictionary:

```
myDictionary.display(sorted_alphabetically=False)   # print the dictionary for
each array index
print(myDictionary)                                 # print the dictionary in
the classical Python fashion
```

The `display` method of the `dictionary` class, with the input argument `sorted_alphabetically` set to `False` (the default), is useful if the user wants to observe the key-value pairs at each non-empty index, as well as the collisions – if there are any. Otherwise, using `print` is sufficient to display the elements of the dictionary in the classical Pythonic way.

> The `display` method only accepts an explicit `True` or `False` input argument. Any other input type will result in a value error.

Printing the alphabetically-sorted dictionary

To print the alphabetically-sorted dictionary, use the `display` method with `sorted_alphabetically` set to `True`:

```
myDictionary.display(sorted_alphabetically=True)    # print the alphabetically-sorted dictionary
```

In the code, to sort the dictionary alphabetically, all of the dictionary keys are used to create `key` objects, which are then appended to a separate array. A `key` object consists of two attributes: the original key, and the same key converted to a string. The string attribute is used to sort the array in an alphabetical manner, while the original one is used to present the alphabetically-sorted dictionary contents with their original data types preserved.

As an example, this is what the array of `key` objects would look like for the dictionary presented at the beginning of this README:

This approach has two main advantages:

1. There is no need to convert the keys from their original type to strings, and then back to the original type. The keys are only converted once to strings, and then sorted alphabetically.
2. It simplifies the handling of the case when the key given by the user is a numeric string (such as `"20"`).

After the objects are sorted, the original keys are used to retrieve the corresponding values, and then the sorted key-value pairs are presented to the user.

> In the alphabetically-sorted dictionary, numeric data comes before the "pure" string data, and uppercase letters come before the lowercase ones.

Additional Information

The project is organized in three files:

- `main.py` – Run this file in order to see the result in your terminal.
- `data_structures.py` – This file contains the necessary data structures and methods needed to complete this project.
- `utils.py` – This file contains the hash function.